E-ISSN: 2583-9667
Indexed Journal
Peer Reviewed Journal
https://multiresearchjournal.theviews.in

Received: 11-07-2025
Accepted: 19-09-2025
Published: 27-10-2025

INTERNATIONAL JOURNAL OF ADVANCE RESEARCH IN MULTIDISCIPLINARY

Volume 3; Issue 4; 2025; Page No. 241-245

Social Network Analysis Using Graph Theory: Centrality Measures, Community Detection, And Telecommunication Network Applications

[1]Madhvi Dhopte and [2]Dr. Akhilesh Kumar Dwivedi

[1]Research Scholar, Mahakaushal University, Jabalpur, Madhya Pradesh, India
[2]Associate Professor, Mahakaushal University, Jabalpur, Madhya Pradesh, India

DOI: https://doi.org/10.5281/zenodo.20676527

Corresponding Author: Madhvi Dhopte

Abstract

The telecoms industry faces intense competition, making it crucial for operators to understand customer habits and preferences. Graph theory and social network analytics can help operators cluster nodes in a large telecommunication network graph. Graph theory's techniques are effectively used in social network analysis, which focuses on networks and their properties. Graph theory essentially sees social networks as mathematical structures comprised of nodes (individuals) and edges (connections or interactions).

Keywords: Graph, telecommunication, network and analytics

Introduction

The mathematical procedure known as SNA is useful for determining social structure. In order to comprehend efficient social structure, this novel method integrates several societal nodes and edges. In addition, several methods have contributed to our understanding of societal patterns and dynamics, including algorithms, statistics, python code, and graph theory. Nodes and edges, which together depict social structure, are the most crucial parts of a SNA. According to (2021), nodes often stand in for people's self-proportions, including their size, weight, location, and other characteristics that contribute to the formation of social networks.

In addition, the neighbors, degree number, network-based features include, among other things, nodes that are cluster-connected and help build the social network of a civilization. However, edges have been essential in establishing connections between nodes (2020). Whereas edges stand for the weight of the individuals and the asymmetry of their relationships within society, edges also stand for the strength of the connections between them. The real social structure may also be better understood with the use of time. All of these things have contributed to a better understanding of how society works. This is where the many human relationships-those involving virtual routing, roads, social connections, physical electricity, biology, and many more-have contributed to the growth of social networks.

Networks that show the relationships between individuals in a graph format for various types of research are called social networks. An example of a graph that may be used to represent human connections is the sociogram. A matrix data structure known as Socio-matrix stores all of the points and lines in the graph. The connections may be of any kind, including familial, friendly, adversarial, professional, social, neighborhood, disease-transmitting, etc.

A social network analysis (SNA) is a method that uses graph theory to examine social networks. Evaluating and assessing the structural features of the network is its principal purpose. Flows and interactions between entities, such as groups and organizations, may be better measured in this way. Researching and analyzing social networks requires specific instruments.

The field of graph theory offers a wealth of mathematical tools for investigating social networks. As an example, notable features such as small-world phenomena or power-law distributions may be shown via the distribution of degrees, which measures how often nodes with various degrees appear. Algorithms for detecting communities based on graph theory may help find smaller communities or strong social relationships by locating groups or clusters within a computer network.

One way to find out which nodes or prominent people in a social network are really controlling the flow of information, influence, or control is to use centrality metrics like degree centrality, betweenness centrality, and eigenvector centrality. Graph algorithms may help us understand how ideas or behaviours spread in social networks such as breadth-first search or random walks to describe diffusion processes.

Literature Review

Truong, Quoc-Dinh et al. (2016) [1]. Nodes in a social network are groups of people who are involved in a certain social activity and are linked to one another via various kinds of interactions. Weighted, labeled, directed graphs are the de facto representation of social networks due to the inherent intricacy of the participants and their interrelationships.

Kar, Rupsha. (2022) [2]. Relationships are fundamental to marketing, and recent studies have expanded our knowledge of relationships beyond basic dyadic ones to think about how social media networks influence people's purchasing decisions. Being able to see how social media can provide an ordinary person a massive platform to connect with people all over the globe. It's like seeing a small-time celebrity rise from the ashes and gain hundreds, if not millions, of followers.

N, Mamatha et al. (2024) [3]. Relationships and interactions in complex systems, such as organizational hierarchies and social media platforms, may be better understood via the use of SNA, or social network analysis. With the mathematical underpinnings provided by graph theory, SNA is able to depict as nodes in a network and the interactions between them as edges. This article explores the basics of graph theory with the purpose of analysing the static and dynamic aspects of social networks. We discuss key graph metrics such as degree centrality, betweenness, closeness, and eigenvector centrality in order to identify significant nodes and communities.

Goldenberg, Dmitri. (2019) [4]. As a field of research, social network analysis applies graph theory and networks to the examination of social systems. Theories that aim to explain the patterns and dynamics seen in social networks are integrated with various approaches for investigating their architecture. It sprang out of social psychology, statistics, and graph theory, but is intrinsically multidisciplinary. This presentation will go over the basics of analysing social networks, including a brief overview of graph theory and data transfer.

Lanel, G.H.J. et al. (2020) [5]. Using ideas from graph theory, this paper makes an effort to describe the interconnected social media networks that include Facebook, Twitter, and LinkedIn. Part one of the research involves cataloguing the aforementioned networks' activities and building a proper graph model to depict such activities on each social media platform. Part two of this research will focus on assessing the built models of each social media platform independently to find out how well they can evaluate user behavior and traits. The last section of this research aims to offer a way for third parties to get access to useful data from online social networks in a way that respects users' privacy and allows for informed decision-making.

Research Methodology

Graph theory and social network analytics will form the basis of our model, which integrates the directed and undirected features of customer call patterns. Clustering models classify users as influencers-those who can persuade their friends to transfer operators-and create client groups based on these characteristics.

Stanford's Network Analysis Platform (SNAP) are used to evaluate the merits of these systems.

ANALYSIS

A library for network analysis and graph mining, Stanford Network Analysis Platform (SNAP) serves a variety of purposes. Written in C++, it effortlessly grows to enormous networks with billions of edges and hundreds of millions of nodes. It supports characteristics on nodes and edges, rapidly produces regular and random graphs, computes structural features, and efficiently manipulates big graphs. SNAP's programming approach is likewise centered on graphs.

Research at Stanford University into Research into extensive online social and information networks has resulted in the consistent growth of the SNAP library, which has been under active development since 2004.

Python provides an interface for SNAP called snap.py. With Snap.py, you get the speed of SNAP with the adaptability of Python. The Python module Snap.py provides access to the majority of SNAP's features.

Updated on May 12, 2015, Snap.py 1.2 is the most recent version. The Snappy repository at Stanford University has packages for Windows 64-bit, Linux (as CentOS), and Mac OS X.

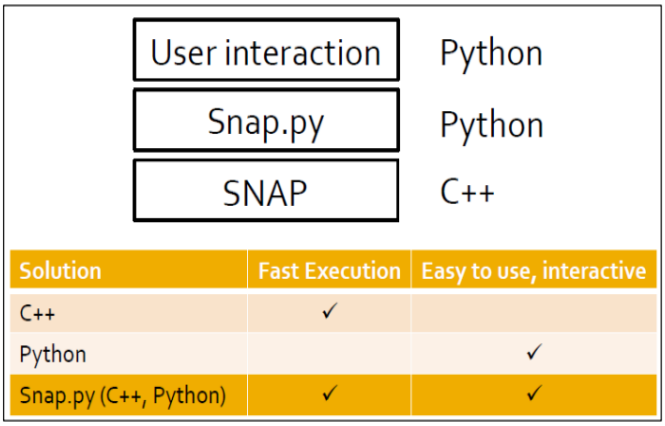

Fig 1: SNAP

Code examples

Add the snap module to your Python environment to access Snap.py.

```
import snap
```

Python automatically converts the basic SNAP types TInt, TFlt, and TStr to int, float, and str, respectively, inside Snap.py. Using SNAP has become more easier because to the automatic conversion types is often unnecessary with Snap.py.

```
i = snap.TInt(10)
print i.Val
```

snap.TInt(10): Create an integer with value 10.
i.Val: Return the number of variable i.

Sequences of items that have a common type are called vectors. You may access or modify existing vector values by looking up their index in the series. A vector may have additional values appended at its end. In Snap.py and SNAP, vector types are typically denoted with followed by V. A vector of integers, for instance, might be called TIntV.

```
v = snap.TIntV()
v.Add(1)
v.Add(2)
v.Add(3)
print v.Len()
print v[2]
v.SetVal(2,6)
print v[2]
```

snap.TIntV(): Create an empty vector of integers
v.Add(1): Add a value at the end of a vector
v.Len(): Get the number of values in the vector
v[2]: Get a value at a specific vector location
v.SetVal(2,6): Change a value at a specific vector location

Values are contained in pairs. Every value is assigned a unique type. The naming pattern for pair types in Snap.py and SNAP is, with Pr following. One example is the TIntStrPr pair, which consists of an integer and a string. One possible application of the type name TIntPr would be if and were of the same type. Pairs might be represented by a vector. Consider the vector TIntPrV, which consists of pairs of integers and integers.

```
p = snap.TIntStrPr(1, "one")
print p.GetVal1()
print p.GetVal2()
```

snap.TIntStrPr(1, "one"): Create a pair of an integer and a string
p.GetVal1(): Print the first value
p.GetVal2(): Print the second value

One edge per line may be imported into a directed, undirected, or multigraph in SNAP using the LoadEdgeList (GraphType, InFNm, SrcColId, DstColId) command.
The graph type that was loaded is indicated by GraphType:

- Directed graph: PNGraph
- Multi graph: PNEANet
- Punctuated graph: an undirected graph

This is an example of a whitespace-separated multi-column file: <source node id>~<destination node id>.
By providing SrcColId and DstColId, you may identify which row represents the edge's origin and which row represents its destination, accordingly.

```
Graph= snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
Graph= snap.LoadEdgeList(snap.PNGraph,"edges.txt",0,1)
```

snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1):
Load the edges of file 'edges.txt' as Undirected graph
snap.LoadEdgeList(snap.PNGraph,"edges.txt",0,1):
Load the edges of file 'edges.txt' as Directed graph

Iterators traverse nodes and edges. Just to illustrate

```
for NI in Graph.Nodes():
    print "nodeId: %d" % (NI.GetId())
for EI in Graph.Edges():
    print "edge (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

Graph.Nodes(): Traverse all the nodes using a node iterator
NI.GetId(): Return the id of the node
Graph.Edges(): Traverse all the edges using an edge iterator
EI.GetSrcNId(): Return the id of the source edge
EI.GetDstNId(): Return the id of the destination edge

Level of Centrality
Absolute centering The total number of links that lead to a node is its in-degree.

SnapInDegreeCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PNGraph,"edges.txt",0,1)
for NI in Graph.Nodes():
    x = str(NI.GetId()) + '\t' + str(NI.GetInDeg())
    print x
```

NI.GetInDeg(): Returns the In-Degree of a node

Out-Degree centrality
The total number of connections that a node forms with other nodes in the network is called its out-degree.

SnapOutDegreeCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PNGraph,"edges.txt",0,1)
for NI in Graph.Nodes():
    x = str(NI.GetId()) + '\t' + str(NI.GetOutDeg())
    print x
```

NI.GetOutDeg(): Returns the Out-Degree of a node

Level of importance
A node's degree of centrality is proportional to the amount of connections that go through it.
SnapDegreeCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PNGraph,"edges.txt",0,1)
for NI in Graph.Nodes():
    x = str(NI.GetId())+'\t'+ str(NI.GetOutDeg()+ NI.GetInDeg())
    print x
```

The Focus on Closeness
The average length of a node's path from origin to every other node in the network shortest route connecting all of its neighbors. Total distance from all nodes decreases as a node's centrality increases.
SnapClosenessCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
for NI in Graph.Nodes():
    closenessValue = snap.GetClosenessCentr(Graph, NI.GetId())
    x = str(NI.GetId()) + '\t' + str(closenessValue)
    print x
```

snap.GetClosenessCentr(Graph, NI.GetId()):
Returns the Closeness centrality of a node, given the Graph as first parameter and id of the node as second parameter

Betweenness Centrality

After tallying up all the shortest routes that go past a certain vertex, we may determine its Betweenness centrality.

SnapBetweennessCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
Nodes = snap.TIntFltH()
Edges = snap.TIntPrFltH()
snap.GetBetweennessCentr(Graph, Nodes, Edges, 1.0)
for node in Nodes:
    x = str(node) + "\t" + str(Nodes[node])
    print x
```

snap.TIntFltH(): Hash table mapping node ids to their corresponding betweenness centrality values.
snap.TIntPrFltH(): Hash table mapping edges (provided as pairs of node ids) to their corresponding betweenness centrality values.
snap.GetBetweennessCentr(Graph, Nodes, Edges, 1.0):
Returns the Betweenness centrality of every node in the network. The last parameter used as indicator of quality of the approximation. Value 1.0 gives the exact Betweenness centrality.

Eigenvector Centrality

A metric known as Eigenvector centrality proves that a vertex's significance is dictated by the significance of its neighbors.

SnapEigenvectorCentrality.py

```
import snap

Graph = snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
NIdEigenH = snap.TIntFltH()
snap.GetEigenVectorCentr(Graph, NIdEigenH)
for item in NIdEigenH:
    x = str(item) + '\t' + str(NIdEigenH[item])
    print x
```

snap.TIntFltH(): Hash table mapping node ids to their corresponding betweenness centrality values.
snap.GetEigenVectorCentr(Graph, NIdEigenH):
Returns the Eigenvector centrality of every node in the network.

Relative Clustering Power

The Clustering Coefficient measures how often clusters develop among nodes in a certain network. Specifically, a user's clustering coefficient is the probability that any two neighbours selected at random are connected.

SnapClusteringCoefficient.py

```
import snap

Graph = snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
NIdCCfH = snap.TIntFltH()
snap.GetNodeClustCf(Graph, NIdCCfH)
for NI in NIdCCfH:
    x = str(NI) + "\t" + str(NIdCCfH[NI])
    print x
```

snap.TIntFltH():
Hash table mapping node ids to their corresponding betweenness centrality values.
snap.GetNodeClustCf(Graph, NIdCCfH):
Returns the Clustering Coefficient of every node in the network.

The K-Core

Complex graph's K-Core Subgraph G is maximally linked if and only if all of the edges in G have degrees greater than or equal to k. Removing all vertices with degrees less than k generates a subgraph of G, which includes this connected section.

SnapKCore.py

```
import snap

Graph = snap.LoadEdgeList(snap.PUNGraph,"edges.txt",0,1)
CoreIDSzV = snap.TIntPrV()
kValue = snap.GetKCoreNodes(Graph, CoreIDSzV)
for item in CoreIDSzV:
    x = str(item.GetVal1()) + "\t" + str(item.GetVal2())
    print x
```

snap.TIntPrV():
Vector of (int, int) pairs.
snap.GetKCoreNodes(Graph, CoreIDSzV):
Returns the number of nodes for a given k-core.

Conclusion

There is a vast array of fields and applications that can benefit greatly from social network analytics that make use of graph theory. Although the results of this study might provide the groundwork for further investigations into social network analytics, it's crucial to remember that any use case for SNA calls for in-depth familiarity with the specific social and graph properties of the target dataset such that a reliable prediction model may be built. The importance processing graphs systems is growing as the number of issues using graphs increases. We were able to learn more about the unique characteristics of each graph processing system by comparing the three systems with data sets that varied in size and characteristics. The association between research cooperation and productivity has also been investigated, and there has been much study on the centrality measures of social networks.

References

1. Truong QD, Dkaki T, Truong QB. Graph methods for social network analysis. In: Proceedings/Book Chapter; c2016. p.276-286. doi:10.1007/978-3-319-46909-6_25.
2. Kar R. To study the impact of social network analysis on social media marketing using graph theory. International Journal of Software Science and Computational Intelligence. 2022;14:1-20. doi:10.4018/IJSSCI.304437.
3. Mamatha N, Sunitha SS, Shivakumar D. Graph theory and its role in social network analysis. World Journal of Advanced Research and Reviews. 2024;21:2088-2093. doi:10.30574/wjarr.2024.21.2.0249.
4. Goldenberg D. Social network analysis: From graph theory to applications with Python. 2019. doi:10.13140/RG.2.2.29075.30244.
5. Lanel GHJ, Jayawardena H. A study on graph theory properties of online social networks. International Journal of Scientific and Research Publications. 2020;10:p9929. doi:10.29322/IJSRP.10.03.2020.p9929.
6. Devineni S, Gorantla B. Graph theory and algorithms for social network analysis. Computer Science Engineering and Technology. 2024;3:52-60. doi:10.46632/jdaai/3/1/7.
7. Kolomeets M, Chechulin A, Kotenko I. Social networks analysis by graph algorithms on the example of the VKontakte social network. Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications. 2019;10:55-75. doi:10.22667/JOWUA.2019.06.30.055.
8. Phule S. Applications of graph theory in networking and social media. International Journal of Advanced Research in Science, Communication and Technology.

2024:466-472. doi:10.48175/IJARSCT-15778.
9. Tripathi A, Gaur A, Sri S. Implementation and analysis of social network graph in interpersonal network. Journal Ilmu Komputer. 2020;13(2):5. doi:10.24843/JIK.2020.v13.i02.p03.
10. Ngo-Hoang DL. Social network analysis: An overview. 2022. doi:10.5281/zenodo.7090093.
11. Kothimbire M, Shelke D, Gaikwad M, Yelpale A, Shinde R. A comprehensive review of graph theory applications in network analysis. International Journal of Mathematics and Computer Research. 2025;13. doi:10.47191/ijmcr/v13i3.08.
12. Vega-Muñoz A, Arjona-Fuentes J. Social networks and graph theory in the search for distant knowledge. In: Social Networks and Knowledge Management. 2020. doi:10.4018/978-1-5225-9380-5.ch017.
13. Arul S, Senthil G, Jayasudha S, Alkhayyat A, Azam K, Elangovan R. Graph theory and algorithms for network analysis. E3S Web of Conferences. 2023;399:08002. doi:10.1051/e3sconf/202339908002.
14. Devi M, Kasireddy S. Graph analysis and visualization of social network big data. In: Advances in Intelligent Systems and Computing. 2019. doi:10.1007/978-981-13-1456-8_8.
15. Shokeen J, Yadav P. Overview of social network analysis and tools. 2016.