Study notes on CSCI 5V75 Special Topics: Text social network analysis

Lou Foua

University of Central Arkansas, Conway AR, USA

1 Introduction

After learning about sentiment analysis and manipulation, I was led into finding how information flows from one person to another. That is where social networks analysis comes. It allows us to study social interactions through graphs.

To accomplish this work, we use NetworkX, which is a Python library, as the software tool. Our main dataset used is a graph that depicts the relationship between some characters of the Game of Thrones book. It summarizes everything we learn in this paper. That dataset comes from a research on Social Network analysis done by Dima Goldenberg [1].

In this study note, I will go through all the concepts related to the analysis of a social network.

2 Social networks

2.1 What is a social network?

According to Merriam-Webster dictionary, a social network is defined as 1: a network of individuals (such as friends, acquaintances, and coworkers) connected by interpersonal relationships, or 2: an online service or site through which people create and maintain interpersonal relationships. In this study, we focus on online apps that let people connect with each other on a common platform.

2.2 How do we represent a social network?

Social networks are commonly represented with graphs. Let's briefly review some basic concepts in graph theory. **A graph** $G(V, E)$ consists a set of **vertices or nodes** V, and a set of edges E. **Edges** are the lines that connect two nodes directly. Fig. 1 illustrates nodes (or vertices) and edges in a sample graph.

A graph can be directed or undirected. **A directed graph**, or **digraph**, is a graph whose edges have directions between two nodes. It is represented with edges that have arrows. **An undirected graph** is a graph whose edges have no arrows. It can go in any direction between two nodes. In a **weighted graph**, each edge is associated with a value. **An unweighted** one is one whose edges don't have any values or score associated with them. Let's look at the images for better illustrations. While Fig. 1 is an undirected and unweighted graph, Fig. 3

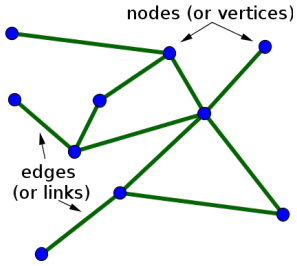

Fig. 1: Nodes and edges in a sample graph.

Fig. 2: Visual showing how social network is represented as a graph

is a directed weighted graph. An undirect graph can be weighted as shown in Fig. 4. In which, each edge is associated with a value without an arrow. In Fig. 5 , there is the summary of all the types of graphs mentioned.

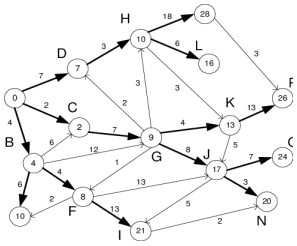

Fig. 3: A directed and weighted graph.

In this study, we use NetworkX to build our network. NetworkX is a python library that allows the creation, manipulation and study of structure dynamics, and functions of complex networks. With this as a software tool, we can represent a social network nicely. Fig. 6 provides sample graphs generated with NetworkX

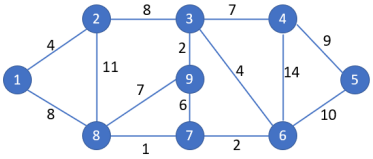

Fig. 4: An undirected and weighted graph.

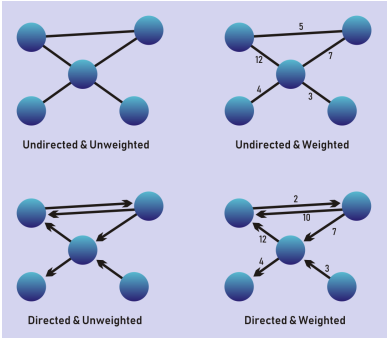

Fig. 5: A summary of the different types of graphs.

3 Types of social networks

In the following subsections, we will see the different types of social networks and their representations with a python code. The library NetworkX has some built-in functions that can build multiple types of graphs. We will describe what each function does as we are using them.

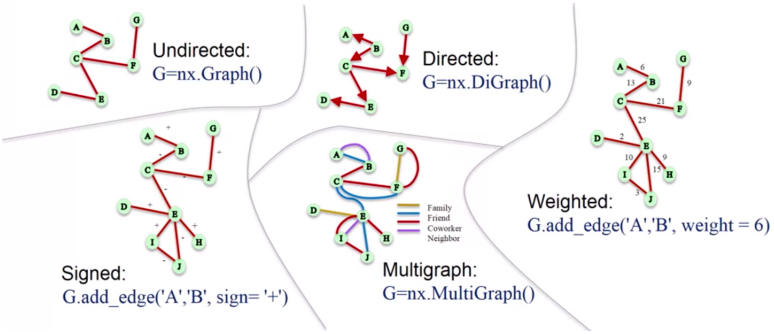

Fig. 6: Graphs generated with NetworkX.

3.1 Small-world network

The idea of a small-word network comes from the concept that people are closer than what we think they are. We can define the concept as the following.

Definition 1. *A small-world network is a type of social network in which all the nodes have a relationship (are connected) with each other in a relatively short distance.*

This basically means that finding a path from one node to another will result in short distance to some extent. This concept is well comprehended through its two major characteristics, that are a high average clustering coefficient and a short average shortest path length. [9]

A clustering coefficient, in graph theory, according to Wikipedia, is a measure of the degree to which nodes in a graph tend to cluster together. In that same vein, it is blatant that creating a "small-world" network in any type of social networks will infer that the nodes will tend to group together, where a high average (or overall) coefficient.

The short average shortest path length is a consequence of the first characteristic. If nodes assemble group effortlessly, that will generate efficiently shortest paths for every node.

Small-world networks illustrate connections seen in roads maps, food chains, electric power grids, metabolite processing networks, networks for brain neurons, voter networks, cultural networks, word co-occurence networks. There is a lot more of the usage of small-world networks. It ranges from telecommunication, transport, social, artificial intelligence to biology. The following is an image of a small world network.

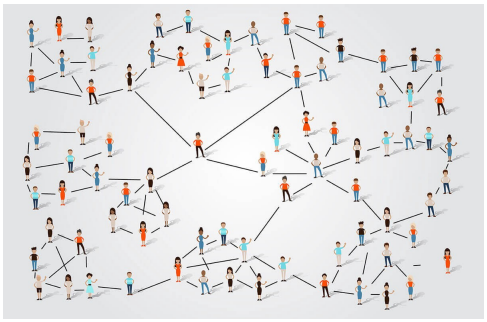

Fig. 7: Illustration of a small world network

In this python code, we are using the connected_watts_strogatz_graph graph, which is a Networkx built-in function to represent a small world graph. Here is a code segment in Python, which generates a small world network with NetworkX [5].

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.connected_watts_strogatz_graph(n=10, k=4, p=0.5, seed=20)
pos = nx.circular_layout(G)
plt.figure(figsize = (12, 12))
nx.draw_networkx(G, pos)
```

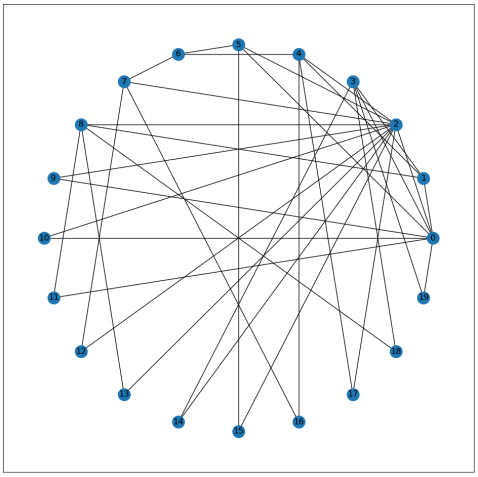

Fig. 8: Result of the Python code segment

3.2 Scale free

Like the small-world network, the scale free network depicts an aspect our world.

Definition 2. *A scale free network is network where the connections among the vertices stand on the power law theory.*

The **power law** is a statistical notion that depicts the dependency of two entities from each other [12]. A change in the value of one entity will result in the change of the value of the other entity proportionally and accordingly to the change made in the first entity. A basic example will be the perimeter of a circle and its radius. If the radius increases, then the perimeter will increase accordingly. There are many other power law relationships we can find in that very geometrical figure. In simpler words, a scale free network is network in which a few nodes have a high a number of connection, while the majority of the nodes does not.

A common notion in that type of network is "hub". In a scale free diagram, the highest degree nodes are called "hubs", even though, as we saw in the previous paragraph, that their notoriety depends on the other nodes.

Scale free networks represent a variety of social, biological, mathematical world examples. For instance, designing the relationships between movie actors or between influencers and followers, will result in a scale free network. Some connections within the brain and metabolic interactions would be expressed by scale free networks. As the example with the circle, a lot of others mathematical problems can illustrated with this kind of social network.

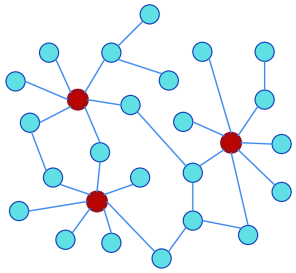

Fig. 9: Illustration of a scale-free network

Looking at a way to get the representation of that type of network, we have the `barabasi_albert_graph` function from NetworkX to represent it. That function creates a random graph and connect the nodes taking their respective degrees into account.

The `barabasi_albert_graph` graph function has the following parameters according to the NetworkX documentation:

- n being the number of nodes
- m the number of edges being attached to new nodes
- seed is the indicator of a random state generator
- the initial graph that has to be transformed into the scale free graph. If there is none, then this parameter will be initial_graph=None

Here is a code segment that will generate a scale free network using the `barabasi_albert_graph` function.

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.barabasi_albert_graph(n = 20, m = 2, seed=10374196,
                    initial_graph = None)
pos = nx.circular_layout(G)
plt.figure(figsize = (12, 12))
nx.draw_networkx(G, pos)
```

3.3 Random network

This type of network illustrates connections that can be made randomly.

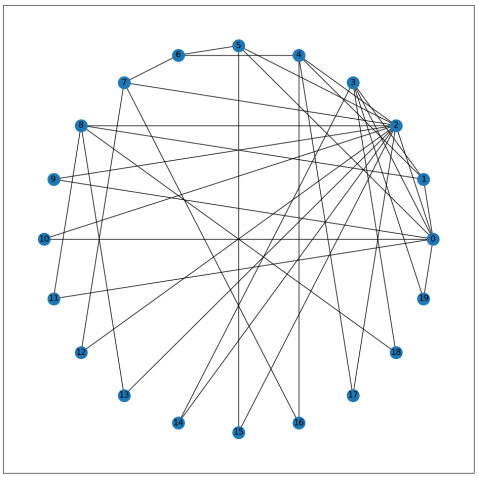

Fig. 10: Illustration of a scale-free network with the result of the Python code segment

Definition 3. *A random network is a type of network that does not follow, neither inspires any pattern.*

Its name transparently says it all. Fundamentally, it is an aleatory network, built without an intended method or design. This network portrays the stochasticity of life. Its purpose stands in its name. We operate with random network whenever we want to display connections made without prearrangement or that does not follow any specific concept. In the following images, you will the see how random networks can be presented.

Random networks are primarily used as benchmarks for other networks. They allow to assess whether a network illustrates randomness or an actual concept. On top of that, they come handy when studying world cases expressing an aleatory pattern. For instance, a random network could be a social media study on how people stumble upon other people's posts whom they have no specific connection with; how connections are made randomly.

To build a random network, we use a NetworkX function called gnp_random_graph. Here are the parameters:

- n being the number of nodes
- p the probability that an edge will be created. That value is float.
- seed is the indicator of a random state generator
- directed that tells if the graph will be directed or not. The default value is directed=False, so the graph is not directed by default. If you want it directed, you can do directed = True

The following lines show a Python code to build a random network with the gnp_random_graph function. [3]

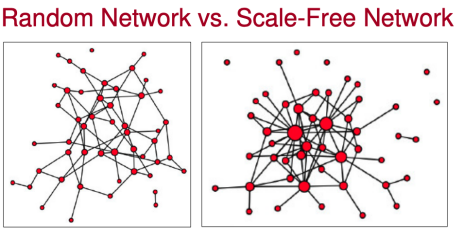

Fig. 11: Visual show how random network can be used as benchmark for scale-free network

```
import matplotlib.pyplot as plt
import networkx as nx
from networkx.generators.degree_seq import expected_degree_graph

N,P = 10, 0.5
G = nx.generators.random_graphs.gnp_random_graph(N, P)

pos = nx.circular_layout(G)

plt.figure(figsize = (12, 12))
nx.draw_networkx(G, pos)
```

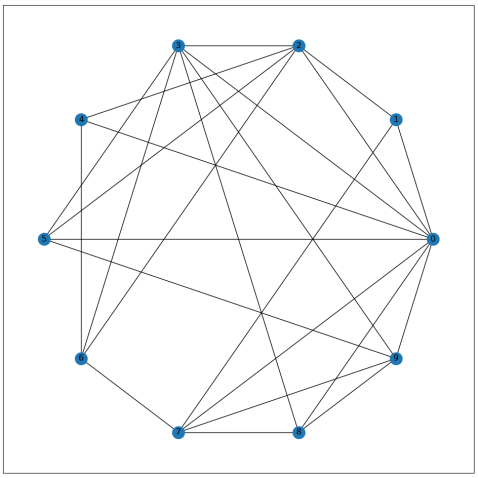

Fig. 12: Visual show the result of the Python code

3.4 Homophily

Definition 4. *A Homophily network is the type of network where the connections among the vertices are determined by their similarities or acquaintances.*

This type of system is used to illustrate likeness in a world. According to wikipedia, homophily in social relations may lead to a commensurate distance in networks leading to the creation of clusters that have been observed in social networking services. For instance, a 16-year-old-girl will more likely connect to a person of her age than to a random 50-year-old-lady, unless they are related family-wise or any other way. This type of network shows the relationship between people that are linked based on a setting or category. This categorization could be based on the age, the school, the family, friends' group, country or whatever other thing that could attach people together.

According to Wikipedia, homophily is a key topic in network science as it can determine the speed of the diffusion of information and ideas.

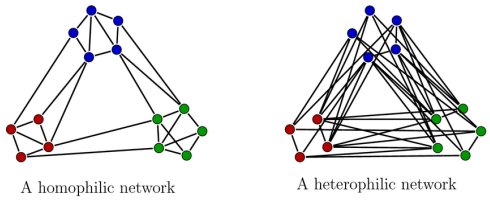

Fig. 13: Visual show the difference between a homophilic and a heterophilic networks

The ego_graph function from NetworkX, builds an homophilic graph. It does so by combining, nodes that are related to each other, around a certain radius. Here are the parameters presented in the NetworkX documentation for this function:

- G being a NetworkX graph or Digraph
- n being a single node
- radius is a number and it is optional. It includes all neighbors of distance that are less than radius from n
- center is an optional, boolean value. If set to False, it does not include the center node in the graph.
- undirected is a boolean and optional value. If True use both in- and out-neighbors of directed graphs
- distance is and optional key that could be used to measure the distance from the node n

Here is the code segment to represent a homophilic network.

```
import networkx for graph generation
import networkx as nx
import matplotlib library
import matplotlib.pyplot as plt
# generation of a sample graph
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'),
                  ('B', 'C'), ('E', 'F'),
                  ('D', 'E'), ('A', 'D'),
                  ('D', 'G'), ('C', 'F'),
                  ('D', 'F'), ('E', 'H')])
# Defining ego as large and red
# while alters are in lavender
# Let 'A' be the ego
ego = 'A'
pos = nx.spring_layout(G)
nx.draw(G, pos, node_color = "lavender",
        node_size = 800, with_labels = True)
# create ego network
hub_ego = nx.ego_graph(G, ego)
# showing the ego network
nx.draw(hub_ego, pos, node_color="lavender",
        node_size = 800, with_labels = True)
nx.draw_networkx_nodes( hub_ego, pos)
plt.show()
```

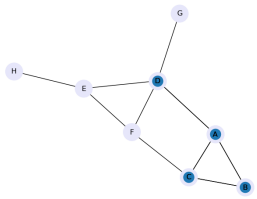

Fig. 14: Visual representing a homophilic network, result of the given Python code

To study a network, we need to understand how it works. And an important aspect of a social network we need to go over is influence. We all have different influences on the people around. We impact some people more than others or we are more impacted by some people than others. To measure the importance of your influence on others, we have this concept of centrality. Centrality will give a score for every element of the network to know its importance. Let's dive into it!

4 CENTRALITY

We have different centrality measures. In a social network, we have a lot of different nodes. Processing all of them can be a hard work, so in the optic of understanding the importance of each of them, we use centrality measures.

Definition 5. *Centralities are metrics used to measure the importance, the relevance or influence of a node in regard to the other nodes in a network.*

There are so many centrality measures that we cannot study all of them together. Here, we will study Degree, Eigen Vector, PageRank, Closeness, Betweenness centralities.

The centrality measures of a node are computed relatively to the node position in the network and its connectivity to the other nodes. Also, using a centrality is based on the purpose of your study, the features you want to look at, and the use of that centrality in the study.

4.1 Degree

In graph theory, a degree of a node is the number of direct edges that node has.

Definition 6. *The degree centrality is the measure of the number of immediate links of a node to its neighbours.*

Having a high degree centrality is interpreted sometimes as being a popular node in the network. In the case of directed graphs, we can define two types of degree centrality that are indegree and outdegree.

An indegree centrality will be based on the edges that are directed to the node we are analyzing.

An outdegree centrality is based on the edges going from that node.

Degree centrality serves for a variety of reasons. On social media, for instance, it can be used for a study users interactions through their number of posts, comments, or contacts. On top of that, degree centrality can be useful in cases like disease contamination or information flow. In the case of directed graphs with indegree and outdegree notions, a high indegree score shows that you have good popularity in the network. However, an oudegree score might show you are a big follower, which can pejorative depending on the study case.

Here is an equation that defines the degree centrality mathematically.

$$C_D(n) = deg(n) \tag{1}$$

with

- C_D stands for degree centrality here in our example
- n n will represent a single node
- $deg(n)$ that is a common function in graph theory to talk about the number of direct connections (edges) a node (n) has

4.2 Betweenness

In a graph, there exists usually a good amount of different paths from a node to another. Obviously, one of these paths will be the shortest (if there is no tie). The betweenness centrality points those nodes that appear in the shortest paths the most.

Definition 7. *The betweenness centrality evaluates the amount of times a node allows a path between two other nodes to be smaller.*

That node is being like an intermediary, a bridge. According to Medium, it tells how much a node helps in information flow in the network . Note that we are stating that the paths go through that node. Therefore the node, whose betweenness centrality we are calculating, should not be one of the end node in a path, it should be in the path.

Betweenness centrality is broadly used in network analysis. It assesses the importance of a node in conveying messages to different ends. Talking about the use of the betweenness centrality, Wikipedia gave a good example. In a telecommunications network, a node with higher betweenness centrality would have more control over the network, because more information is passing through that node.

The betweenness centrality of a node v is given by the expression:

$$C_B(n) = \sum_{s \neq n \neq t} \frac{\sigma_s t(n)}{\sigma_s t} \tag{2}$$

with

- C_B is the betweenness centrality function
- n n represents a single node
- s s represents a node
- t n represents another node
- $\sigma_s t(n)\sigma_s t$ is the total number of shortest paths from node s to node t, and $\sigma_s t(n)$ is the total number of shortest paths from s to t that goes through n.

4.3 Closeness

The closeness centrality is similar to the betweenness one in the sense that both look at the shortest paths.

Definition 8. *The closeness centrality gauges the nodes to point out the ones that have the highest number of shortest paths.*

Closeness centrality analyzes the reciprocity of a node to the other ones in a network. It measures the efficiency with which a node is capable of communicating with others [10]. If a node has the highest number of short paths, that means it is the closest to everything in the network. In fact, a vertex with a high closeness centrality will be the most central vertex in the network. It could be

assimilated to the core of the system. This picture summarizes everything that has said about closeness centrality.

However, as for the degree and betweenness centralities, everything becomes different when it comes to undirected graphs. Similarly to them, there are two types of centralities: outbound closeness and inbound closeness centralities.

An outbound closeness centrality of a node is about the all the connections going from the node. In that case, having a score means that the node is highly solicited. For instance, in a study of a variety of restaurants and their customers, the customer that lives the closest to those restaurants will have the highest outbound closeness centrality.

An inbound closeness centrality of a node concerns all the connections coming to the node. In that same example given before, the closest restaurant to their customers will have a high inbound closeness centrality.

Closeness centrality is used for a number of reasons. It is used in communication network to find the nodes that most important in broadcasting information in websites or on social media. In biology, closeness centrality helps recognize the genes or proteins that key roles in metabolic processes. In transportation, for instance, it is used to identify to the hub of the stations in different locations. [11]

Here a mathematic representation of the closeness centrality.

$$C_C(n) = \frac{N-1}{\sum_u d(u,n)} \tag{3}$$

with

- C_C stands for closeness centrality here in our example
- n n represents a single node
- N is the number of nodes in the graph
- u is any singular other node
- $d(u,v)$ is the distance (shortest distance) between the node n and a certain node u in the network
- $\sum_u d(u,v)$ is the sum of all the distance from the node n to all the other nodes u in the network

4.4 Eigen vector

The Eigen centrality goes a step further than to the degree one. It not only counts the number of direct connections, but it also extends to all the other nodes. It counts the number of edges to get to the other nodes. Eigen vector pulls out complex relationships in a network. It informs about the connections that are made, their importance.

Definition 9. *Eigenvector centrality gauges the importance of nodes and their connections in a social network.*

It is a more complex, but more accurate evaluation of the worth of a node in the network. For instance, a given node can be pointed at by many other nodes, but

if those nodes don't have a high eigen vector centrality score, then that given node will not have a high one neither. However, if a given node is pointed at by a lot other nodes that have high eigenvector centralities, then that given will have a high centrality as well. Eigenvector centrality works in the details.

The history of the eigenvector centrality goes back to 1895 with a man called Edmund Landau. Landau was using it to score chess tournaments. In our recents years, researchers have explored the use of the Eigen vector in a wide range of fields. Here are some cited by Wikipedia.

- Eigenvector centrality is the unique measure satisfying certain natural axioms for a ranking system.
- In neuroscience, the eigenvector centrality of a neuron in a model neural network has been found to correlate with its relative firing rate.
- Eigenvector centrality and related concepts have been used to model opinion influence in sociology and economics, as in the DeGroot learning model.
- In a study using data from the Philippines, researchers showed how political candidates' families had disproportionately high eigenvector centrality in local intermarriage networks.
- Eigenvector centrality has been extensively applied to study economic outcomes, including cooperation in social networks. In economic public goods problems, a person's eigenvector centrality can be interpreted as how much that person's preferences influence an efficient social outcome.

4.5 Pagerank

PageRank is very similar to Eigen vector, but with more calculations. Similarly to the Eigen vector, it takes the number of edges of a node to all the other nodes. Its particularity is that, it adds the weight of those nodes or edges and their directions in the computation.

Definition 10. *PageRank is the Eigen vector centrality for directed graphs.*

Initially, PageRank (PR) was an algorithm that was used by Google Search to rank web pages in their search engine results. Its names comes from a combination of the term "web page" and the co-founder Larry Page. PageRank helps measure the significance of a page. Here is how Google states that: PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites. Although it is no longer used at Google, PageRank is known as the first algorithm used by Google to order search results.

Obviously, pagerank is used in web search. In social network analysis, it is used in to identify people that have the most impact on the public opinion or a certain group of people. In academics, biometrics, business or finance, it shows influence, connectivity between elements in a meaningful and relevant way.

This image shows all the types of centrality that were discussed above.

Now that we have talked about centrality and illustrated with images, let's see what it will look like in a code. [2]

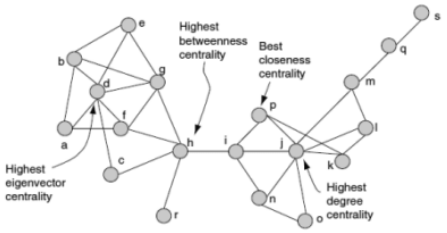

Fig. 15: Visual showing the nodes with the highest score in every centrality type

```
import matplotlib.pyplot as plt
import networkx as nx

G = nx.karate_club_graph()

plt.figure(figsize =(15, 15))
nx.draw_networkx(G, with_labels = True)

#Degree Centrality
deg_centrality = nx.degree_centrality(G)

# G is the Karate Club Graph
print("\n\nDegree Centrality: ", deg_centrality)
#in case the graph is directed, this is what has to be used
#in_deg_centrality = nx.in_degree_centrality(G)
#out_deg_centrality = nx.out_degree_centrality(G)

#Closeness Centrality
close_centrality = nx.closeness_centrality(G)
# G is the Karate Social Graph
print("\n\nCloseness Centrality: ", close_centrality)

#Betweeness Centrality
bet_centrality = nx.betweenness_centrality(G, normalized = True,
                                           endpoints = False)
# G is the Karate Social Graph, parameters normalized
# and endpoints ensure whether we normalize the value
# and consider the endpoints respectively.
print("\n\nBetweeness Centrality: ", bet_centrality)


#PageRank Centrality
pr = nx.pagerank(G, alpha = 0.8)
# G is the Karate Social Graph
```

```
print("\n\nPageRank Centrality: ", pr)

#Eigen Vector Centrality
eigen = nx.eigenvector_centrality(G)
print("\n\nDEigen vector Centrality: ", eigen)
```

Here is the result yielded:

```
Degree Centrality:  {0: 0.48484848484848486, 1: 0.2727272727272727,
2: 0.30303030303030304, 3: 0.18181818181818182, 4: 0.09090909090909091,
5: 0.12121212121212122, 6: 0.12121212121212122, 7: 0.12121212121212122,
8: 0.15151515151515152, 9: 0.06060606060606061, 10: 0.09090909090909091,
11: 0.030303030303030304, 12: 0.06060606060606061, 13: 0.15151515151515152,
14: 0.06060606060606061, 15: 0.06060606060606061, 16: 0.06060606060606061,
17: 0.06060606060606061, 18: 0.06060606060606061, 19: 0.09090909090909091,
20: 0.06060606060606061, 21: 0.06060606060606061, 22: 0.06060606060606061,
23: 0.15151515151515152, 24: 0.09090909090909091, 25: 0.09090909090909091,
26: 0.06060606060606061, 27: 0.12121212121212122, 28: 0.09090909090909091,
29: 0.12121212121212122, 30: 0.12121212121212122, 31: 0.18181818181818182,
32: 0.36363636363636365, 33: 0.5151515151515151}

Closeness Centrality:
{0: 0.5689655172413793, 1: 0.4852941176470588, 2: 0.559322033898305,
3: 0.4647887323943662, 4: 0.3793103448275862, 5: 0.38372093023255816,
6: 0.38372093023255816, 7: 0.44, 8: 0.515625, 9: 0.4342105263157895,
10: 0.3793103448275862, 11: 0.36666666666666664, 12: 0.3707865168539326,
13: 0.515625, 14: 0.3707865168539326, 15: 0.3707865168539326, 16: 0.28448275862068967,
17: 0.375, 18: 0.3707865168539326, 19: 0.5, 20: 0.3707865168539326, 21: 0.375,
22: 0.3707865168539326, 23: 0.39285714285714285, 24: 0.375, 25: 0.375,
26: 0.3626373626373626, 27: 0.4583333333333333, 28: 0.4520547945205479,
29: 0.38372093023255816, 30: 0.4583333333333333, 31: 0.5409836065573771,
32: 0.515625, 33: 0.55}

Betweeness Centrality:  {0: 0.43763528138528146, 1: 0.053936688311688304,
2: 0.14365680615680618, 3: 0.011909271284271283, 4: 0.0006313131313131313,
5: 0.02998737373737374, 6: 0.029987373737373736, 7: 0.0, 8: 0.05592682780182781,
9: 0.0008477633477633478, 10: 0.0006313131313131313, 11: 0.0, 12: 0.0,
13: 0.04586339586339586, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.0, 19: 0.03247504810004811,
20: 0.0, 21: 0.0, 22: 0.0, 23: 0.017613636363636363, 24: 0.0022095959595959595,
25: 0.0038404882154882154, 26: 0.0, 27: 0.02233345358345358, 28: 0.0017947330447330447,
29: 0.0029220779220779218, 30: 0.014411976911976909, 31: 0.13827561327561325,
32: 0.145247113997114, 33: 0.30407497594997596}

PageRank Centrality:  {0: 0.08681447719172829, 1: 0.05576990180502326,
2: 0.06022609788477227, 3: 0.03670727111555245, 4: 0.021184995124117666,
5: 0.034116513708762286, 6: 0.031875010542794575, 7: 0.026078126588955267,
8: 0.032504029578824645, 9: 0.01050443442853472, 10: 0.021394066044403352,
11: 0.010843321085063444, 12: 0.012430356457194331, 13: 0.03255335020695045,
14: 0.013723097706681172, 15: 0.016885155427766807, 16: 0.017615950148322918,
17: 0.010728169791778151, 18: 0.010603973010401791, 19: 0.013847703406997999,
```

20: 0.012142068846138356, 21: 0.012266674546455183, 22: 0.013744564219084298,
23: 0.04005234377538896, 24: 0.017273255096315228, 25: 0.02841630372234375,
26: 0.016038313813914594, 27: 0.027151493976771628, 28: 0.015095117409551478,
29: 0.028413382290443677, 30: 0.02296983977569958, 31: 0.041092418400124585,
32: 0.07407774615757656, 33: 0.09486047671556612}

Eigen vector Centrality: {0: 0.35548349418519426, 1: 0.2659538704545024,
2: 0.3171893899684447, 3: 0.21117407832057056, 4: 0.0759664588165738,
5: 0.07948057788594245, 6: 0.07948057788594245,7: 0.1709551149803543,
8: 0.22740509147166046, 9: 0.10267519030637756, 10: 0.0759664588165738,
11: 0.05285416945233646, 12: 0.08425192086558085, 13: 0.22646969838808145,
14: 0.10140627846270832, 15: 0.10140627846270832, 16: 0.02363479426059687,
17: 0.0923967566684595, 18: 0.10140627846270832, 19: 0.14791134007618667,
20: 0.10140627846270832, 21: 0.0923967566684595, 22: 0.10140627846270832,
23: 0.15012328691726787, 24: 0.057053735638028055, 25: 0.0592082025027901,
26: 0.07558192219009324, 27: 0.13347932684333308, 28: 0.13107925627221215,
29: 0.13496528673866567, 30: 0.17476027834493088, 31: 0.191036269797917,
32: 0.3086510477336959, 33: 0.37337121301323506}

The result obtained is a dictionary with the nodes as keys and their scores as values. The node that has the biggest is the one that leads in centrality in the different types given above. For instance, in the Eigen Vector Centrality, the 4 most important nodes, in descending order are the 33, 1, 32 and 2.

Now we have gone over the different types of degrees, let's see how we can cluster them using those degrees.

5 NETWORK INFLUENCERS FOR CLUSTERING

In that section, we will see how we can use our knowledge of centrality to make clusters. Clusters are usually highly influenced by certain nodes called **network influencers**. Those influencers are determined by the different centralities measures we have studied before. The following subsections are examples of clusters made based on the different types of centralities. All of them use the same data source, but yield independent results. The data source is named was constructed by the Standford University and can be downloaded at this following reference [1]. Given the example of betweenness, and using NetworkX, the process of building clusters will be the following:

- We create a graph from the data source using NetworkX. This will like this in Python

```
G_fb = nx.read_edgelist("facebook_combined.txt",
 create_using = nx.Graph(), nodetype=int)
```

- Then we use colors and sizes to make the networks influencers stand out from the other nodes of the network. In coding, it will like something like

[1] https://snap.stanford.edu/data/egonets-Facebook.html

this: creating two variables color and size, and assigning a color to node depending on its centrality value(its influence in the network). Basically, it will look like this in code

```
node_color = [10000 * v for v in betCent.values()]
node_size =  [v * 10000 for v in betCent.values()]
```

The following subsections will show the Python code, and the result to build clusters based on different types of centrality.

5.1 Betweenness centrality

```
import networkx as nx
import matplotlib.pyplot as plt

G_fb = nx.read_edgelist("facebook_combined.txt", create_using = nx.Graph(),
                        nodetype=int)
pos = nx.spring_layout(G_fb)
betCent = nx.betweenness_centrality(G_fb, normalized=True, endpoints=True)
node_color = [10000 * v for v in betCent.values()]
node_size =  [v * 10000 for v in betCent.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G_fb, pos=pos, with_labels=False, node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(betCent, key=betCent.get, reverse=True)[:5]
```

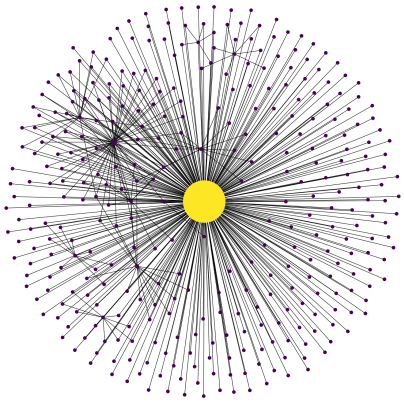

Fig. 16

On the above graph Fig. 16, the bold-filled-yellow-circle in the middle represents the node with the highest value of betweenness centrality in the network. It

is a really central node. Most of the nodes in the network go through this yellow node to reach others. And that is why the yellow one is so bold and big, because it is like a bridge between a lot of other nodes. In the case someone is studying the flow of information in a network, and wants to define who will the one that has the most will know everything, it will be the yellow bold circle. Besides, you can see some other nodes that created some clusters. You can count around 8 small clusters, with important nodes, but less important than the yellow one.

5.2 Eigen vector centrality

```
import networkx as nx
import matplotlib.pyplot as plt

G_fb = nx.read_edgelist("facebook_combined.txt", create_using = nx.Graph(),
                                nodetype=int)
pos = nx.spring_layout(G_fb)
centr = nx.eigenvector_centrality(G_fb, max_iter=100, tol=1e-06,
                            nstart=None, weight=None)
node_color = [10000 * v for v in centr.values()]
_size =  [v * 10000 for v in centr.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G_fb, pos=pos, with_labels=False, node_color=node_color,
                 node_size=node_size )
plt.axis('off')
```

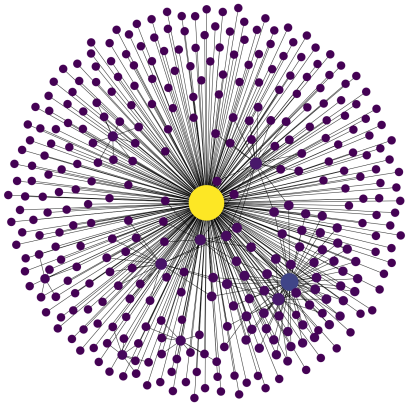

Fig. 17

The above figure Fig. 17 shows clusters made with the Eigen vector centrality. The bold yellow-filled circle is the node with the biggest Eigen centrality, followed

by the light purple one and then the dark purple ones. We can represent the whole network as a cluster with the yellow one bubble, and then smaller clusters with the other colors as the image shows.

5.3 Degree centrality

```
import networkx as nx
import matplotlib.pyplot as plt

G_f b = nx.read_edgelist("facebook_combined.txt",
                         create_using = nx.Graph(), nodetype=int)


pos = nx.spring_layout(G_fb)
centr = nx.degree_centrality(G_fb)
node_color = [10000 * v for v in centr.values()]
node_size =  [v * 10000 for v in centr.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G_fb, pos=pos, with_labels=False, node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(centr, key=centr.get, reverse=True)[:5]
```

The above figure Fig. 18 shows clusters made with the degree vector centrality. The bold yellow-filled circle is the node with the biggest Eigen centrality, followed by the light purple one. Then, we have the big dark purple ones, and lastly the small purple ones.

5.4 PageRank centrality

```
import networkx as nx
import matplotlib.pyplot as plt

G_fb = nx.read_edgelist("facebook_combined.txt", create_using = nx.Graph(),
                                    nodetype=int)
pos = nx.spring_layout(G_fb)
centr = nx.pagerank(G_fb, alpha=0.85, personalization=None, max_iter=100,
                    tol=1e-06, nstart=None, weight='weight', dangling=None)
node_color = [10000 * v for v in centr.values()]
node_size =  [v * 10000 for v in centr.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G_fb, pos=pos, with_labels=False, node_color=node_color,
                 node_size=node_size )
plt.axis('off')
```

The above figure Fig. 19 shows clusters made with the pagerank centrality. The bold yellow-filled circle is the node with the biggest Eigen centrality, followed

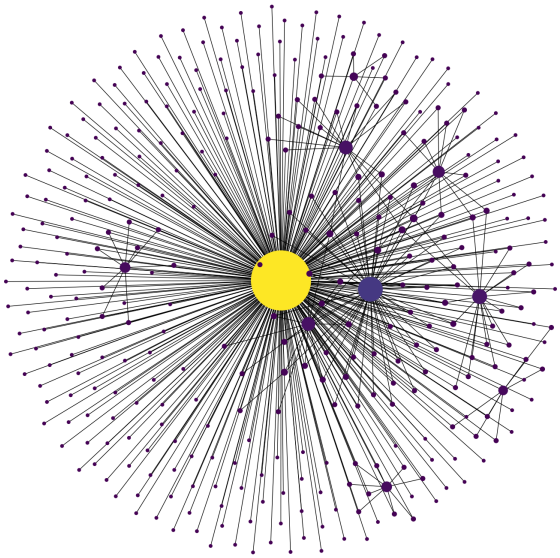

Fig. 18

by the light purple one and lastly the dark purple ones from the biggest size to the smallest one.

5.5 Closeness centrality

```
import networkx as nx
import matplotlib.pyplot as plt

G_fb = nx.read_edgelist("facebook_combined.txt", create_using = nx.Graph(),
                                    nodetype=int)
pos = nx.spring_layout(G_fb)
centr = nx.closeness_centrality(G_fb, u=None, distance=None,
                                              wf_improved=True)
node_color = [10000 * v for v in centr.values()]
node_size =  [v * 10000 for v in centr.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G_fb, pos=pos, with_labels=False,
                     node_color=node_color, node_size=node_size )
plt.axis('off')
```

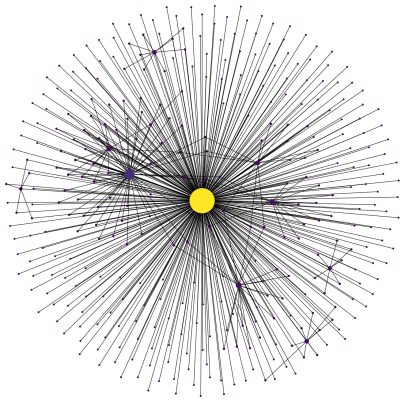

Fig. 19

The above figure Fig. 20 shows clusters made with the closeness centrality. The bold yellow-filled circle is the node that is the closest to the other. It looks all the nodes in the network are close to each other.

6 INFORMATION FLOW

Understanding information flow in social networks has important applications in viral marketing, public health campaigns, and mitigating the spread of misinformation. In the following subsections, we will discover different information diffusion models.

6.1 Linear threshold

The **linear threshold model** is a widely studied framework in social network analysis to understand information flow, influence propagation, or adoption of behaviors. This model is especially relevant in understanding how collective behaviors or trends spread in social systems. [14]

Definition 11. *The linear threshold model depicts how individuals in a network decide to adopt a new idea, behavior, or innovation based on the influence of their neighbors after reaching a certain threshold. [13]*

Some important concepts are to be taken into account: the threshold, the nodes states, the edges states and weights. **The threshold** represents the value to which a node is activated, or affected by what has been coming to it. Each node has a personal threshold which represents the fraction of influence it needs from its neighbors to adopt the behavior. For example, if a node might require at

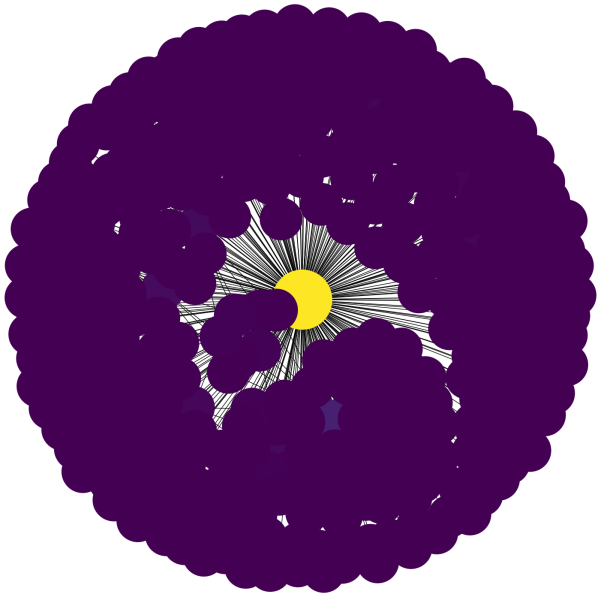

Fig. 20

least 30 percent of its neighbors to adopt the behavior before it does the same. That is its threshold. **A node state** basically shows whether a node is activated or not. **The edges state** is the direction of the edges. In the linear threshold information flow diffusion, connection not only matters, but the direction of the connection matters the most. **The weights on edges** are just normal values as we have seen earlier with the weighted graph in the different types of graphs sections. Each directed edge has a weight representing the strength of influence from node to node.

Now let's talk about the **activation process**. A node is **active** if it has adopted the behavior or idea, and **inactive** otherwise. At each time step, a node becomes active if the total influence from its active neighbors exceeds its threshold. The process begins with a set of initially active nodes called **seed nodes**. These nodes act as the starting point for the propagation. The seed nodes start as active. All other nodes are inactive. At each step, inactive nodes evaluate the influence from their active neighbors. Nodes exceeding their threshold become active and can influence their neighbors in the next step. The process continues until no new nodes become active, leading to a steady state

Linear threshold as diffusion model can be applied to a wide range of fields. It is used in marketing and viral campaigns for identifying influential individuals (seed nodes) to maximize the spread of a product or idea. Besides, it can be used in epidemiology to model the spread of information about diseases or

health behaviors. Moreover, we see its use in political campaigns. It helps with studying how political messages or ideologies spread within communities. On top of that, there is network robustness assessment that can benefit from the linear threshold model in understanding how resistant a network is to adopting harmful behaviors. [14]

However, linear threshold has some limitations. There is a fixed threshold. The assumption that a node has a certain specific threshold may oversimplify real-world behavior. Additionally, there is a homogeneity of Influence. We know that most real world problem cannot be confined into a linear model. In reality, influence is often dynamic and context-dependent. Furthermore, the accurate edge weights and thresholds are difficult to estimate for real-world networks. [15]

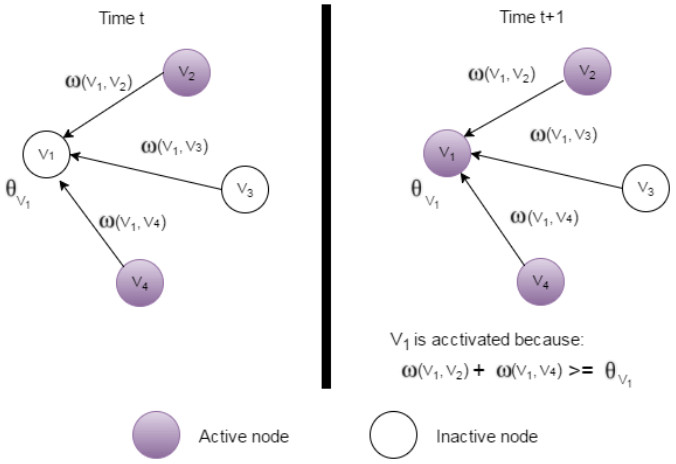

Fig. 21

In the above illustration Fig. 21 , **v2** and **v4** are the seeds nodes. Then **v3** , and **v1** are inactive nodes. **v2** and **v4** activate the node **v1**. You can see on the image the computation for the threshold that results in the activation of **v1**

6.2 Independent cascade

The **independent cascade** (**IC**) model is another widely studied framework for understanding information flow, influence propagation, and behavior adoption in social networks. [16]

Definition 12. *Unlike the linear threshold model, where influence depends on a cumulative threshold, the independent cascade model treats influence as a probabilistic, one-time activation process. [16]*

The key components of the independent cascade model are the nodes and edges states, the probability of influence, and the seed nodes. As of before, the node state is about the activation status of a node. The edge state, however, here, is not about direction. It is about connection and weight (probability here) of that connection between two nodes. The **Probability of Influence** constitute the likelihood that a node will successfully activate another node. That means, we can have a connection between two nodes with one activated and the other not, but there will be no activation for that one because of the probability of influence. The seed nodes, as of before, are a subset of nodes that are initially activated. These nodes initiate the diffusion process. Talking about the activation process, it starts when a node becomes active. The very first nodes to start are, obviously, the seeds nodes. Then, they activate their neighbors based on the following condition. Every node gets a single chance to activate each of its neighbors. The activation attempt is independent for each neighbor and succeeds with the probability of influence of that node on its neighbors. The process continues until no more nodes can be activated. [13]

The independent cascade model is used in a variety of cases. Similarly to the linear threshold model, it helps to understand the spread of diseases and design strategies to limit transmission in epidemiology. In the viral marketing, it identifies influential individuals (seed nodes) to maximize the spread of products or campaigns. Moreover, in the fight against misinformation and fake news, it enables the modeling of false information spreads across networks. [16]

Talking about the limitations of the independent cascade model we have the fact that the parameter estimation in determining realistic edge probabilities for real-world networks is challenging. The model assumes independence between activation attempts, which may not hold true in all real-world scenarios.

In the figure Fig. 22 , we see the propagation of information through independent cascade in the famous movie show Games of Thrones. Here, the seed nodes are "Jon Snow" and "Robert Baretheon". They are already on the process of activation. The white bubbles are activated, the yellow bubbles are nodes determined to be activated and on the process of being so. Finally, the blues ones are inactive nodes. From "Jon Snow" and "Robert Baretheon", the activation continues "Sam Stark" and "Samwell Tarly" and so on and so forth, until there is no more nodes to be activated. That means the activated nodes, do not have a high probability of influence towards their immediate neighbors so activation cannot be done. And it is the end, unless something new happens in the network.

The following table summarizes the features of both models we learned about.

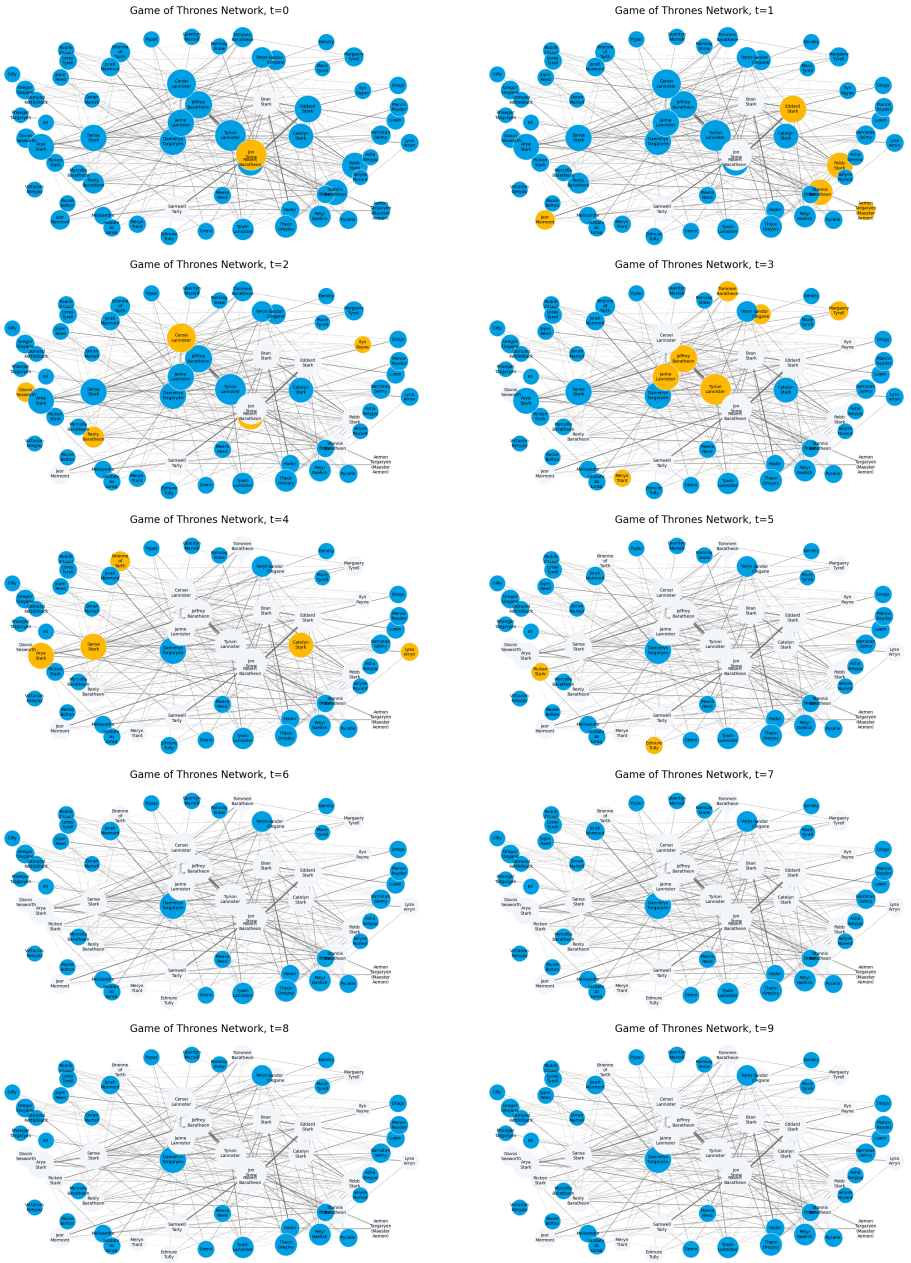

Fig. 22: Independent cascade

Feature	Independent Cascade	iLinear Threshold
Type of Influence	Probabilistic, one-time activation attempt	Deterministic or cumulative influence
Activation Trigger	Random chance (per edge probability)	Cumulative influence reaching a threshold
Focus	Individual interaction probabilities	Collective influence of neighbor
Propagation	Asynchronous, stepwise influence	Simultaneous influence evaluation

7 SUMMARY AND FURTHER STUDY

This report explored the application of social network analysis in understanding the dynamics of relationships within a social network.We have learned the representations of social networks with graphs, and its concepts like centralities.

Then, we learned to use those centralities to cluster elements in a graph. Clusters are made based on the network influencers, which are just the centralities measures. A same graph can yield different types of clustering with different types of centralities measures. Finally, we analyzed the information flow in graph through the individual diffusion models. Linear threshold and independent cascade were the two kinds of information flow patterns we studied.

As we learned that social networks can be used in different fields like physics, chemistry, telecommunications, we can, as a follow-up study, analyze models use cases in a specific field.

References

1. D. Goldenberg: Social Network Analysis: From Graph Theory to Applications with Python, arXiv preprint arXiv:2102.10014, 2021. Last referenced on October 30, 2024.
2. Network Centrality Measures in a Graph using Networkx | Python, https://www.geeksforgeeks.org/network-centrality-measures-in-a-graph-using-networkx-python/
3. How to generate a random network but keep the original node degree using networkx?, https://stackoverflow.com/questions/48321963/how-to-generate-a-random-network-but-keep-the-original-node-degree-using-network
4. Ego graph Using Networkx in Python, https://www.geeksforgeeks.org/ego-graph-using-networkx-in-python/
5. Software for Complex Networks, https://networkx.org/documentation/stable/index.html
6. Types of graphs, https://medium.com/tebs-lab/types-of-graphs-7f3891303ea8
7. Social network definition https://www.techtarget.com/searchcio/definition/social-network

8. Basic network analysis on a directed network using NetworkX, https://waterprogramming.wordpress.com/2021/01/19/basic-network-analysis-on-a-directed-network-using-networkx/
9. Small world networks, https://www.jsums.edu/nmeghanathan/files/2015/08/CSC641-Fall2015-Module-6-Small-World-Networks-reduced.pdf
10. Closeness centrality, https://en.wikipedia.org/wiki/Closeness
11. When closeness centrality algorithm best applied, https://www.graphable.ai/blog/closeness-centrality-algorithm/
12. Scale-free network, https://en.wikipedia.org/wiki/Scale-free
13. D. Kempe, et al, Maximizing the Spread of Influence through a Social Network. https://www.cs.cornell.edu/home/kleinber/kdd03-inf.pdf
14. M. Granovetter. Threshold Models of Collective Behavior. https://www2.cs.siu.edu/ hexmoor/classes/CS539-F10/Collective-Behavior.pdf
15. D. Easley, J. Kleinberg. Networks, Crowds, and Markets. https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf
16. J. GoldenBerg, et al. Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth. https://business.columbia.edu/sites/default/files-efs/pubfiles/3391/TalkofNetworks.pdf