SOCIAL INFLUENCE MAXIMIZATION
USING NATURE-INSPIRED ALGORITHMS

MINOR PROJECT – II

SUBMITTED BY –

HARSHIT SHARMA (9919103016)
YASHA JAFRI (9919103009)
PRANAT JAIN (9919103017)
SIDDHARTH SINGH (9919103029)

UNDER THE SUPERVISION OF –

DR. SHIKHA K MEHTA

DEPARTMENT OF CS/IT
JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

APRIL 2022

ACKNOWLEDGEMENT

We would like to place on record our deep sense of gratitude to Dr. Shikha K Mehta, Associate Professor, Jaypee Institute of Information Technology, India for her generous guidance, help and useful suggestions.

We also wish to extend our heartfelt gratitude to our batchmates for their insightful comments and constructive suggestions to improve the quality of this project work. Lastly, we would like to thank our parents to be the constant source of motivation which we needed during the course of this project.

Harshit Sharma

Harshit Sharma (9919103016)

[illegible]

Siddharth Singh (9919103029)

Pranat Jain

Pranat Jain (9919103017)

Yasha Jafri.

Yasha Jafri (9919103009)

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place: Noida, Uttar Pradesh

Date: 7th April, 2022

Harshit Sharma
9919103016

Yasha Jafri
9919103009

Pranat Jain
9919103017

Siddharth Singh
9919103029

CERTIFICATE

This is to certify that the work titled “Social Influence Maximization using Nature Inspired Algorithms” submitted by Harshit Sharma, Yasha Jafri, Pranat Jain and Siddharth Singh of B.Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree or diploma.

Dr. Shikha K Mehta

(Associate Professor, Jaypee Institute of Information Technology)

7th April, 2022

ABSTRACT

Since the beginning, the biggest trait that has enhanced Human development has been ‘Learning from others. Many of our actions we take today are influenced by others. People learn from each other and this has important implications on a person’s thought processes. The biggest platform where people can influence others' choices and decisions is Social Media.

Due to the growth of the Internet and Web 2.0, many large-scale online social network sites like Facebook, Twitter, Instagram, etc. became successful because they are very effective tools in connecting people and bringing small and disconnected offline social networks together. Moreover, they are also becoming a huge dissemination and marketing platform, allowing information and ideas to influence a large population in a short period of time. However, to fully utilise these social networks as marketing and information dissemination platforms, many challenges have to be met.

In this project, we present our work towards addressing one of the challenges, finding influential individuals/groups efficiently in a large-scale social network. This problem, referred to as *Influence Maximisation* , would be of interest to many companies as well as individuals who want to promote their products, services, and innovative ideas through the powerful word-of-mouth effect (or, called viral marketing). Online social networks provide good opportunities to address this problem, because they are connecting a huge number of people and they collect a huge amount of information about the social network structures and communication dynamics.

Influence Maximisation is the problem of finding a small subset of nodes (seed nodes) in a social network that could maximise the spread of influence. This is an active area of research in the computational social network analysis domain. Due to its practical importance in various domains, such as viral marketing, target advertisement and personalised recommendation, the problem has been studied in different variants, and different solution methodologies have been proposed over the years.

This project mainly focuses on designing an algorithm that identifies such nodes in a social network that maximise influence. We also idealise to improve the efficiency of the algorithm using different approaches and models like Independent Cascade Model, Linear Threshold Model, Discrete Shuffled Frog-leaping Algorithm and Jellyfish Algorithm.

TABLE OF CONTENTS

INTRODUCTION

Since the beginning, the biggest trait that has enhanced Human development has been ‘Learning from others. Many of our actions we take today are influenced by others. People learn from each other and this has important implications on a person’s thought processes. The biggest platform where people can influence others' choices and decisions is Social Media.

Due to the growth of the Internet and Web 2.0, many large-scale online social network sites like Facebook, Twitter, Instagram, etc. became successful because they are very effective tools in connecting people and bringing small and disconnected offline social networks together. Moreover, they are also becoming a huge dissemination and marketing platform, allowing information and ideas to
influence a large population in a short period of time. However, to fully utilise these social networks as marketing and information dissemination platforms, many challenges have to be met.

Influence Maximisation is the problem of finding a small subset of nodes (seed nodes) in a social network that could maximise the spread of influence. This is an active area of research in the computational social network analysis domain. Due to its practical importance in various domains, such as viral marketing, target advertisement and personalised recommendation, the problem has been studied in different variants, and different solution methodologies have been proposed over the years.

BACKGROUND STUDY

WHAT IS SOCIAL INFLUENCE MAXIMZATION?

A social network is an interconnected structure among a group of agents formed for social interactions. Social networks play an important role in spreading information, opinions, ideas, innovations, rumors, etc., at a large scale.

A Social Network is abstracted as a Graph, most popularly used to represent real-world networked systems, with the users as the Vertex Set, Social Ties as Edge Set, Diffusion Threshold (a measurement of how hard it is to influence the user; more the value the harder it is to influence the user, values lie in the range (0,1]) as Vertex Weight, and Influence Probability (the probability that the given user node will be influenced by the user node connected to it by that edge, values lie in the range [0,1]) as Edge Weights.

Consider the case of promoting a brand by of a commercial house through online marketing, where the goal is to attract the users for purchasing a particular product.
The best way to do this is to select a set of highly influential users and distribute them free samples. Many of them will like the item and influence their neighbour's to try the product. These newly informed users will influence their neighbours. This cascading process will be continued, and ultimately a large fraction of the users will try for the product leading to a significant improvement in the earned revenue.

Social Influence Maximization Problem:- refers to the selection of those highly influential node

INDEPENDENT CASCADE MODEL

The Independent Cascade Model is an information diffusion model where the information flows over the network through cascade.
Nodes can have two states
Active: it means the node is already influenced by the information in diffusion
Inactive: node is unaware of the information or not influenced.

The process runs in discrete steps. At the beginning of ICM process, few nodes are given the information, they are known as seed nodes. Upon receiving the information these nodes become active. In each discrete step, an active node tries to influence one of its inactive neighbours. Regardless of its success, the same node will never get another chance to activate the same inactive neighbour. The success of node u in activating the node v depends on the propagation probability of the edge (u, v) defined as puv, each edge has its own value. The process terminates when no further node gets activated.

LINEAR THRESHOLD MODEL

for any node, all its neighbors, who are activated just at the previous timestamp together make a try to activate that node. This activation process will be successful, if the sum of the incoming active neighbor's probability becomes either greater than or equal to the node's threshold . then ui will become active at time stamp t + 1. This method will be continued until no more activation is possible. In this model, we can use the negative influence, which is not possible in IC Model.

SIM Problem and its variants

- top k-node Problem/Social Influence Maximization Problem (SIM Problem)

For a given social network $G(V, E,\theta, P)$, this problem asks to choose a set S of k nodes (i.e., $S \subseteq V(G)$ and $|S| = k$), such that the maximum number of nodes of the network become influenced at the end of diffusion process, i.e., σ (S) will be maximized. Most of the algorithms presented in Sect. 6 are solely developed for solving this problem.

- Basic SIM Problem [1]**:** In the basic version of the TSS Problem along with a directed social network $G(V, E,\theta, P)$, we are given two integers: k and λ and asked to find out a subset of at most k nodes, such that after the diffusion process is over at least λ number of nodes are activated

- Influence Spectrum Problem **:** along with the social network G(V , E,θ, P), we are also given with two integers: klower and kupper with kupper > klower. Our goal is to choose a set S for each k ∈ [klower, kupper], such that social influence in the network (σ (S)) is maximum in each case. Intuitively, solving one instance of this problem is equivalent to solving (kupper −klower +1) instances of SIM Problem. As viral marketing is basically done in different phases and in each phase, seed set of different cardinalities can be used, influence spectrum problem appears in a natural way
- Multi-Round Influence Maximization Problem [114] Most of the existing studies of influence maximization consider that the seed set selection is one shot task, i.e., the entire seed set has to be selected before the diffusion starts. However, in many real-world advertisement scenarios, it may be required that the viral marketing need to be conducted in multiple times. To model this scenario, 'Multi-Round Influence Maximization Problem' has been introduced by Sun et al. [114]. In this problem, along with the social network G(V, E,θ, P), we are also given with two integers: k and T . Here, T is the number of times the diffusion process needs to be conducted and k is the cardinality of the seed set. Here, the goal is to choose the seed nodes S1, S2,..., ST , such that at the end of the entire diffusion process, the total number of influenced nodes becomes maximum.
- Target Set Selection Problem **::** for a given social network G(V, E,θ, P), the goal is to select a seed set, whose initial activation leads to the complete influence in the network, i.e., all the nodes are influenced at the end of diffusion process
- Weighted Target Set Selection Problem **::** Along with a social network G(V, E,θ, P), we are given another vertex weight function, φ : V(G) → N0, signifying the cost associated with each vertex. This problem asks to find out a subset S, which minimizes total selection cost, and also all the nodes will be influenced at the end of diffusion
- Budgeted Influence Maximization Problem **::** Along with a directed graph G(V, E,θ, P), we are given with a cost function C : V(G) −→ Z+ and a fixed budget B ∈ Z+. Cost function C assigns a nonuniform selection cost to every vertex of the network, which is the amount of incentive need to be paid, if that vertex is selected as a seed node. This problem asks for selecting a seed set within the budget, which maximizes the spread of influence in the network.

DISCRETE SHUFFLED FROG-LEAPING ALGORITHM

It is a methodology, which depends on imitation of the behaviour patterns of frogs, taking into account a crowd of frogs leaping in a swamp, on the lookout for the place, which has the highest food quantity reachable, in which the swamp has multiple stones at distinct points that make it easier for the frogs to step on.

The aim is to identify a stone with the maximum food amount available.

Communicating between frogs can progress their memes as infection can be propagated among them. As a result of improvement in memes, each frog's position will be changed by tuning its leaping step size. The population is meant to be a number of solutions which is represented as a swarm of frogs that is segregated into subgroups denoted to as memeplexes.

Each memeplex represents a distinct frog community, each conducting a local quest. The specific frogs grip ideas within each memeplex, which can be persuaded by further frogs' ideas and progress through a memetic evolution process.

In a shuffling process, ideas are passed between memeplexes exactly after several memetic evolution phases.

This algorithm combines the advantages of two categories of genetics-based algorithms (such as memetics) and social behaviour-based algorithms (such as the PSO bird algorithm). It seeks to strike a balance between extensive scrutiny in the space of possible answers. In this population algorithm, a population of frogs (answers) consists, each frog will have a chromosome-like structure in the genetic algorithm. The whole population of frogs is divided into smaller groups, each group representing different types of frogs that are scattered in different places of the answer space. Each group of frogs then begins a precise local search around their habitat.

REQUIREMENT ANALYSIS

The requirements for this project are:

1) Python 3.x
   Python is a very varied tool used for Predictive Analysis, due to the availability of various modules which ease the implementation of the required analyses.

2) Python Modules –
   i) Pandas – to create and manage data frames containing data
   ii) NetworkX – provides tools for implementing Graph Networks in Python
   iii) Matplotlib – Plotting the Network Graph

3) Google Colab (For Implementation purposes)
   It is an Online tool to work on our projects without the need to install any modules on our own environment. It has several advantages over the traditional ways to create a Virtual Environment, where several users ran into problems.

DETAILED DESIGN

1) To work on the Social Influence Maximization, we need a Social Network to work upon. We have used the Game of Thrones Network to start with. The Dataset was created from the famous books of "A Song of Ice and Fire" on which the popular TV Series "Game of Thrones" is based upon. The Dataset counts an undirected edge between two characters if they occur in any 15-word frame within the book. Here, characters are the fictional characters involved in the book. The Dataset used is divided into 5 different datasets which contains the edge counts between two characters in the same book in a single dataset.

2) We use the Pandas Library to import these datasets and concatenate these datasets to form a single data frame to work upon. The resultant Data Frame is converted into another data frame of different structure along 3 columns – Source, Target and Weight, using the 'group_by' functionality of Pandas Library, where the 'Source' is the Source Node, 'Target' is the Target Node and 'Weight' refers to the Edge weight between two nodes.

3) Once we have received a Pandas Edge list, we use the 'from_pandas_edgelist' function of the NetworkX library to create an Undirected Graph which will be used later in the code for various purposes. To visualize the Graph, we also plot this graph using the Matplotlib Library.

4) To start with the DSFLA, we first have to create a Population of Memes, which is created by choosing Random nodes from the graph. The size of population, memeplexes to be created, memes in each memeplex and meme types to be contained in a single meme are taken as input.

5) Once the Population has been created, we calculate the Local Influence Estimation Value (LIE) for each meme to divide these memes into different memeplexes. To calculate the LIE for each meme we use the below given formula –

$$\boxed{\begin{aligned} LIE(S) &= k + \sigma_1^*(S) + \frac{\sigma_1^*(S)}{|N_S^{(1)} \setminus S|} \sum_{u \in N_S^{(2)} \setminus S} p_u^* d_u^* \\ &= k + \left(1 + \frac{1}{|N_S^{(1)} \setminus S|} \sum_{u \in N_S^{(2)} \setminus S} p_u^* d_u^*\right) \\ &\quad \times \sum_{i \in N_S^{(1)} \setminus S} \left(1 - \prod_{(i,j) \in E,\, j \in S} (1 - p_{i,j})\right) \end{aligned}}$$

where $N_S^{(1)}$ and $N_S^{(2)}$ represent the one-hop and two-hop area of candidate set S, respectively. p_u^* is a small constant probability of a propagation cascade model. d_u^* is the number of edges of node u within $N_S^{(1)}$ and $N_S^{(2)}$.

6) We sort the Population of Memes in descending order of their respective LIE Values and use the descending order nodes to create the memeplexes such that there is equal representation of fitness value or LIE Values in the Memeplexes.

7) Once we have created the memeplexes, we iterate over each memeplex and follow the steps (a) to (d) in order –
   a) Construct a Sub memeplex from the existing Memeplex
   b) Choose the P_b and P_w, which represent the Best and Worst Frogs of the Submemeplex
   c) We choose a New Meme with respect to the Local Best meme to replace the Worst Meme and calculate its LIE Value. If the calculated LIE value of the new meme using the Local Best

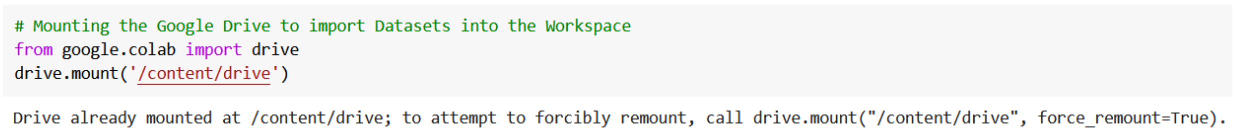
Meme is greater than the current LIE value of the worst meme, we replace the worst meme by the New Meme, otherwise we repeat the same process for the New meme created with respect to Global Best Meme.

d) If the Worst Meme has been replaced by any of the new meme created in the above step, we go to the next step, otherwise we choose a random frog containing random meme types and replace the worst meme.

8) We iterate over the population of memeplexes to improve the fitness of memes in a memeplex.

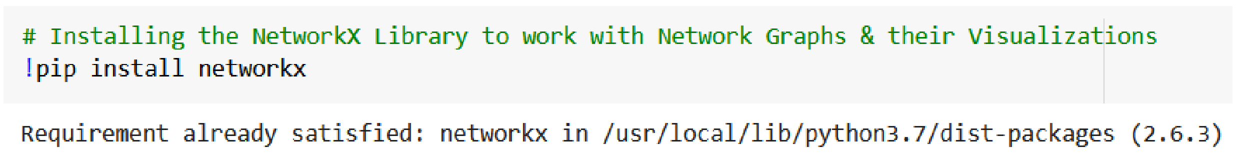
9) We rearrange the memes in the descending order of their LIE values, once the iterations are over. Once the resultant population is achieved, the Meme with the best LIE value is chosen as the Top-K Nodes for the purpose of Influence Maximization.

IMPLEMENTATION



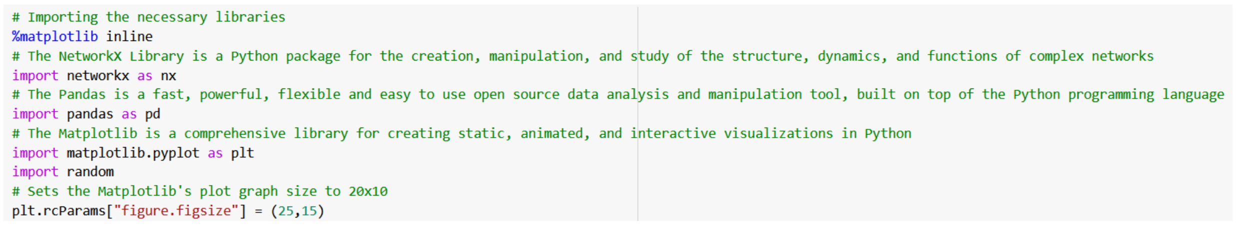
```
# Mounting the Google Drive to import Datasets into the Workspace
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

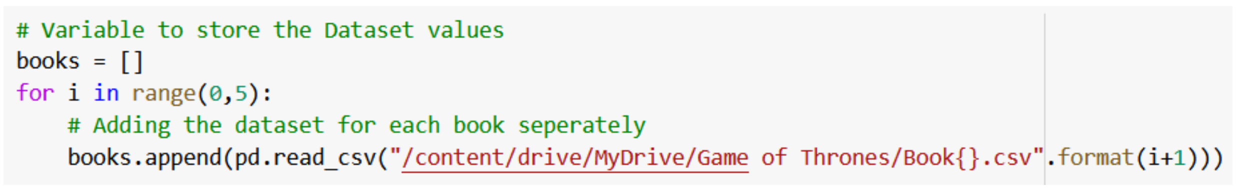


```
# Installing the NetworkX Library to work with Network Graphs & their Visualizations
!pip install networkx
```

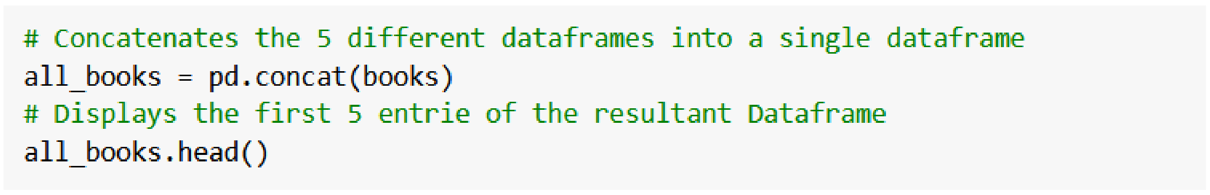
```
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (2.6.3)
```



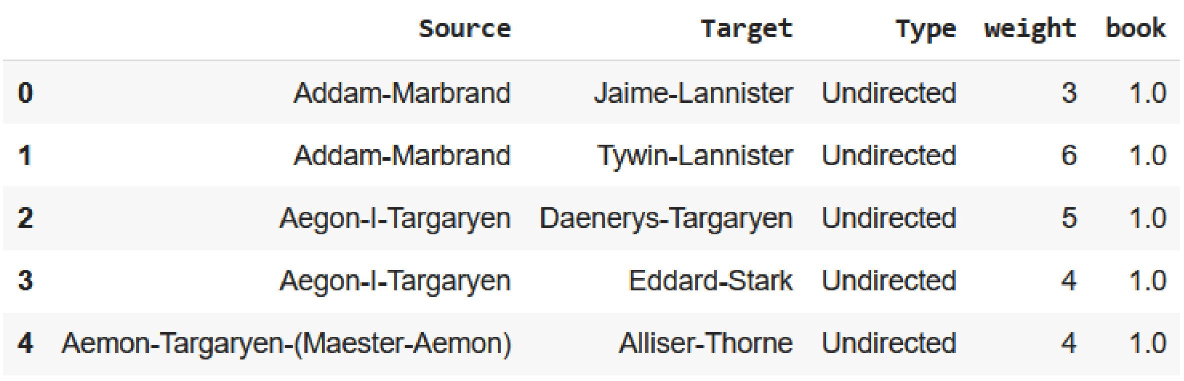
```
# Importing the necessary libraries
%matplotlib inline
# The NetworkX Library is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks
import networkx as nx
# The Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language
import pandas as pd
# The Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
import matplotlib.pyplot as plt
import random
# Sets the Matplotlib's plot graph size to 20x10
plt.rcParams["figure.figsize"] = (25,15)
```



```
# Variable to store the Dataset values
books = []
for i in range(0,5):
    # Adding the dataset for each book seperately
    books.append(pd.read_csv("/content/drive/MyDrive/Game of Thrones/Book{}.csv".format(i+1)))
```



```
# Concatenates the 5 different dataframes into a single dataframe
all_books = pd.concat(books)
# Displays the first 5 entrie of the resultant Dataframe
all_books.head()
```



	Source	Target	Type	weight	book
0	Addam-Marbrand	Jaime-Lannister	Undirected	3	1.0
1	Addam-Marbrand	Tywin-Lannister	Undirected	6	1.0
2	Aegon-I-Targaryen	Daenerys-Targaryen	Undirected	5	1.0
3	Aegon-I-Targaryen	Eddard-Stark	Undirected	4	1.0
4	Aemon-Targaryen-(Maester-Aemon)	Alliser-Thorne	Undirected	4	1.0

```
# Converting the Dataframe structure into an Edgelist to feed into the NetworkX Library function
edges = all_books.groupby(['Source','Target']).agg({'weight':'sum'}).reset_index()
edges.sort_values('weight',ascending=False)
```

	Source	Target	weight
1334	Eddard-Stark	Robert-Baratheon	334
2031	Jon-Snow	Samwell-Tarly	228
1965	Joffrey-Baratheon	Sansa-Stark	222
1972	Joffrey-Baratheon	Tyrion-Lannister	219
640	Bran-Stark	Hodor	209
...	...	...	...
1809	Jaime-Lannister	Joy-Hill	3
1805	Jaime-Lannister	Jon-Snow	3
1802	Jafer-Flowers	Jaremy-Rykker	3
1801	Jaehaerys-I-Targaryen	Maegor-I-Targaryen	3
0	Addam-Marbrand	Brynden-Tully	3

2823 rows × 3 columns

```
# Creating the NetworkX Graph using Pandas Edgelist
GOT = nx.from_pandas_edgelist(edges, source = 'Source', target = 'Target', edge_attr = 'weight')
print(nx.info(GOT))
```

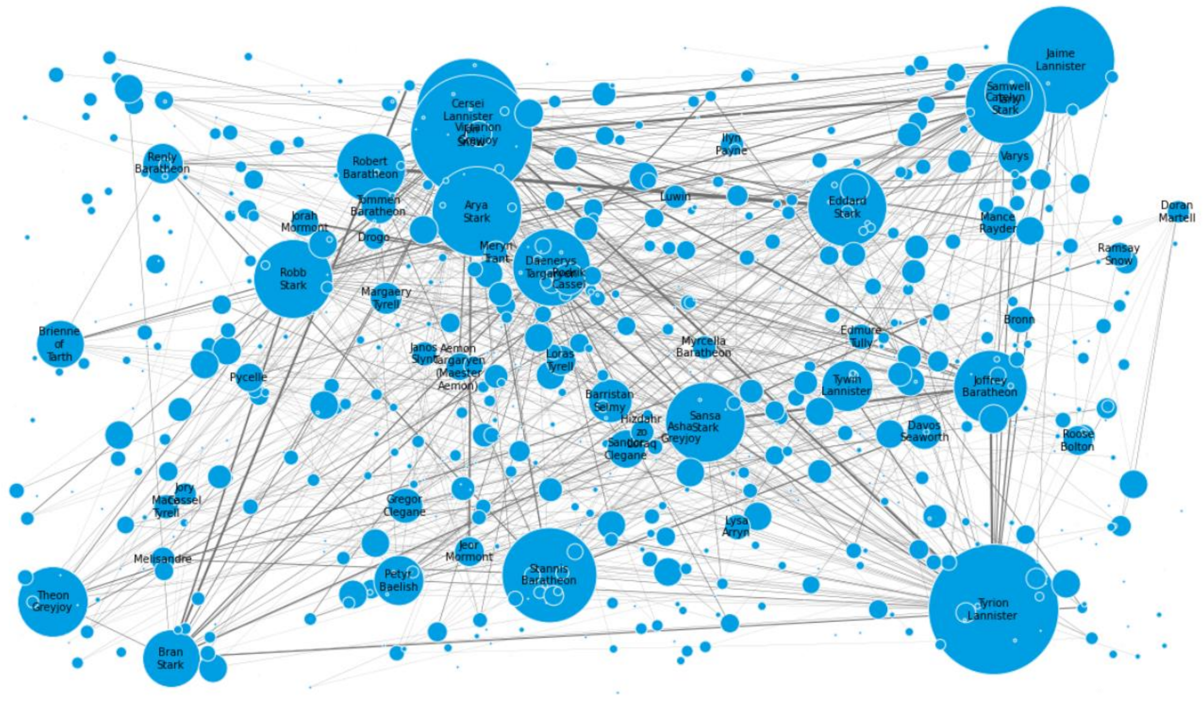
```
Graph with 796 nodes and 2823 edges
```

```
# Assigning Random Probabilities to Nodes for the Constant Propagation Probability of the Cascade Model
nodes = list(GOT.nodes())
node_prob = {}
for i in nodes:
    t = random.randint(0, 100) / 100
    node_prob[i] = t
```

```
# Dictionar storing the Degree of each node in the Graph
weighted_degrees = dict(nx.degree(GOT))
# Max Degree corresponds to the Highest degree of any node in the graph
max_degree = max(weighted_degrees.values())
```

```
# Plotting the Graph Network theoretically
pos = nx.spring_layout(GOT, weight = 'weight', iterations = 1)
```

```
# Plotting the Graph Network on Matplotlib
plt.axis('off')
plt.title('Game of Thrones Network', fontsize = 24)

for node in GOT.nodes():
    if weighted_degrees[node] < 10:
        size = weighted_degrees[node]**3
    else:
        size = weighted_degrees[node]**2
    ns = nx.draw_networkx_nodes(GOT, pos, nodelist = [node], node_size = size, node_color = '#009fe3')
    ns.set_edgecolor('#f2f6fa')

nx.draw_networkx_labels(GOT,pos,{n:n.replace('-','\n') for n in GOT.nodes() if weighted_degrees[n] > 20}, font_size = 10)

for e in GOT.edges(data = True):
    if e[2]['weight'] > 10:
        nx.draw_networkx_edges(GOT, pos, [e], width = e[2]['weight'] / 100, edge_color = '#707070')
```



```
m = int(input("Enter the no. of memeplexes : "))
n = int(input("Enter the no. of memes in each memeplex : "))
k = int(input("Enter the no. of memetypes in each meme : "))
itr = int(input("Enter the no. of iterations to be done : "))
F = m * n
memeplex = []
population = []
for i in range(F):
    population.append([random.choice(list(weighted_degrees)) for i in range(k)])
```

```
Enter the no. of memeplexes : 10
Enter the no. of memes in each memeplex : 10
Enter the no. of memetypes in each meme : 5
Enter the no. of iterations to be done : 20
```

```
for i in range(len(population)):
    print("Meme-{}".format(i+1), population[i])
```

```
Meme-1 ['Ternesio-Terys', 'Craster', 'Widower', 'Dancy', 'Mezzara']
Meme-2 ['Todder', 'Torbert', 'Gerold-Grafton', 'Dick-Crabb', 'Husband']
Meme-3 ['Marwyn', 'Gorne', 'Sandor-Clegane', 'Qyburn', 'Cley-Cerwyn']
Meme-4 ['Willow-Heddle', 'Emrick', 'Grunt', 'Ulf-son-of-Umar', 'Archibald-Yronwood']
Meme-5 ['Mellei', 'Shadrich', 'Morrec', 'Robar-Royce', 'Lark']
Meme-6 ['Aemon-Targaryen-(Maester-Aemon)', 'Lancel-Lannister', 'Conn', 'Heward', 'Bartimus']
Meme-7 ['Rhaella-Targaryen', 'Oznak-zo-Pahl', 'Smiling-Knight', 'Ardrian-Celtigar', 'Lothor-Brune']
Meme-8 ['Prendahl-na-Ghezn', 'Murenmure', 'Ryman-Frey', 'Denzo-Dhan', 'Elmar-Frey']
Meme-9 ['Aenys-Frey', 'Ben-Bones', 'Alys-Karstark', 'Osfryd-Kettleblack', 'Jafer-Flowers']
Meme-10 ['Captain-Myraham', 'Melisandre', 'Richard-Horpe', 'Jon-Arryn', 'Alerie-Hightower']
```

```
def one_hop_area(meme):
    temp = []
    if len(meme) == 1:
        meme = meme[0]
    for i in meme:
        temp.extend(list(GOT.neighbors(i)))
    return list(set(temp) - set(meme))
```

```
def two_hop_area(seed_set, one_hop):
    temp = []
    if len(seed_set) == 1:
        seed_set = seed_set[0]
    for i in one_hop:
        temp.extend(list(GOT.neighbors(i)))
    return list(set(temp) - set(seed_set) - set(one_hop))
```

```
def calc_prob_pcm(group):
    prob = []
    for i in group:
        prob.append(node_prob[i])
    return prob
```

```
def calc_edges(one_hop, two_hop):
    count = []
    for i in two_hop:
        edges = list(nx.edges(GOT, nbunch = [i]))
        temp = 0
        for i in edges:
            if (i[1] in one_hop) or (i[1] in two_hop):
                temp += 1
        count.append(temp)
    return count
```

```python
def sum_pd(pu, du):
    p = 0
    for i in range(len(pu)):
        p += (pu[i] * du[i])
    return p
```

```python
def calc_edge_prob(nodes, S):
    if len(S) == 1:
        S = S[0]
    prob_sum = 0
    for i in nodes:
        temp = list(nx.edges(GOT, nbunch = [i]))
        temp_sum = 0
        for j in temp:
            temp_sum += GOT.get_edge_data(j[0], j[1], default = {'weight':0})['weight']
        prod = 1
        for j in S:
            tprob = GOT.get_edge_data(i, j, default = {'weight':0})
            prod = prod * (1 - (tprob['weight'] / temp_sum))
        prod = 1 - prod
        prob_sum += prod
    return prob_sum
```

```python
def LIE(meme):
    Ns1_S = one_hop_area(meme)
    Ns2_S = two_hop_area(meme, Ns1_S)

    pu = calc_prob_pcm(Ns2_S)
    du = calc_edges(Ns1_S, Ns2_S)
    return k + ((1 + ((1 / len(Ns1_S)) * sum_pd(pu, du))) * calc_edge_prob(Ns1_S, meme))
```

```python
def sort_memes(ppl_lie, ppl):
    for i in range(1, len(ppl_lie)):
        key1, key2 = ppl_lie[i], ppl[i]
        j = i-1
        while (j >= 0) and (key1 > ppl_lie[j]):
            ppl_lie[j + 1] = ppl_lie[j]
            ppl[j + 1] = ppl[j]
            j -= 1
        ppl_lie[j + 1] = key1
        ppl[j + 1] = key2
```

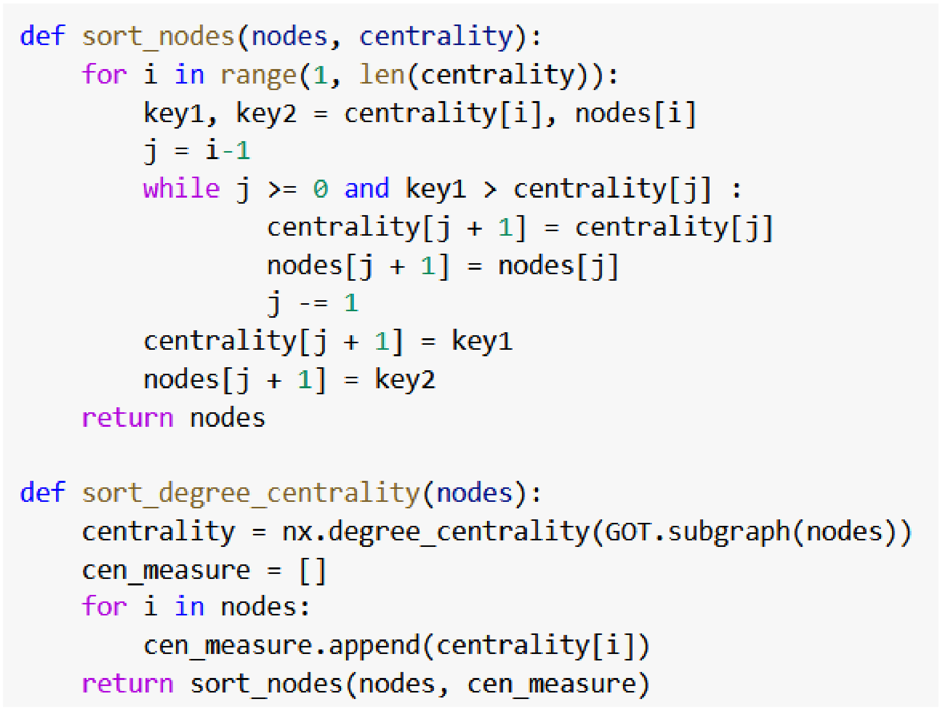
```python
population_LIE = []
for i in range(F):
    population_LIE.append(LIE(population[i]))
sort_memes(population_LIE, population)
```

```
for i in range(len(population)):
    print("Meme: {}".format(i+1), population_LIE[i], population[i])
```

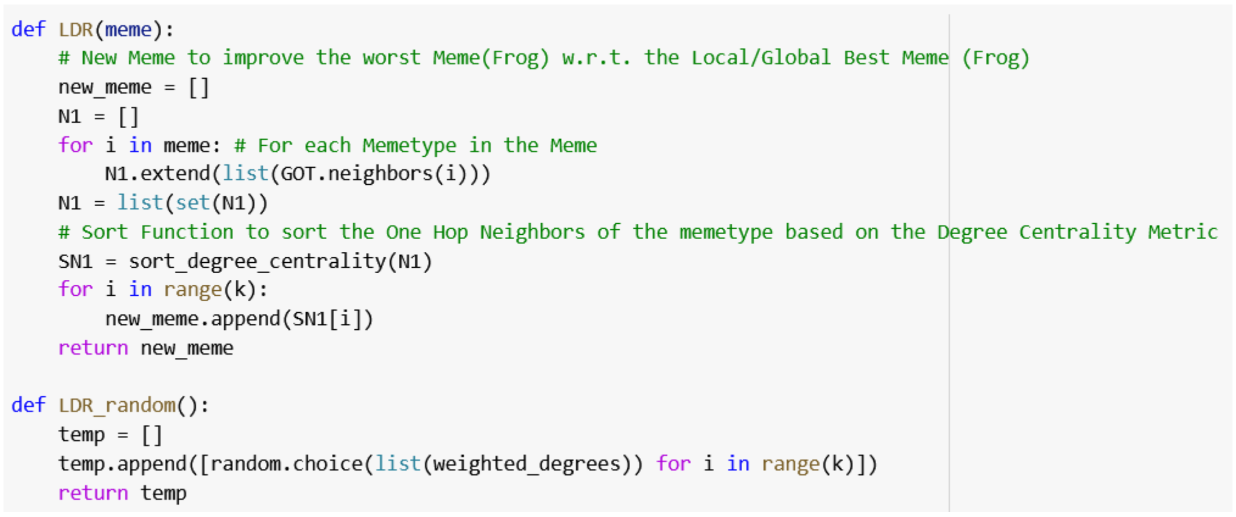
```
Meme: 1 437.83603432746315 ['Xaro-Xhoan-Daxos', 'Daenerys-Targaryen', 'Luwin', 'Orell', 'Jafer-Flowers']
Meme: 2 347.82278870734257 ['Nestor-Royce', 'Gorne', 'Garth-(Wolfs-Den)', 'Tybero-Istarion', 'Bran-Stark']
Meme: 3 343.27250797931686 ['Guncer-Sunglass', 'Meris', 'Oswell-Kettleblack', 'Eddard-Stark', 'Nage']
Meme: 4 308.302842987214 ['Luwin', 'Rus', 'Emmon-Frey', 'Lewyn-Martell', 'Jaime-Lannister']
Meme: 5 299.56584804892793 ['Ygritte', 'Scar', 'Arianne-Martell', 'Alyn', 'Ravella-Swann']
Meme: 6 289.79605828347474 ['Mathis-Rowan', 'Waymar-Royce', 'Hobber-Redwyne', 'Catelyn-Stark', 'Tanda-Stokeworth']
Meme: 7 289.1206036917941 ['Samwell-Tarly', 'Jhogo', 'Osfryd-Kettleblack', 'Waif', 'Cortnay-Penrose']
Meme: 8 265.58684624354396 ['Danwell-Frey', 'Ben-Plumm', 'Lancel-Lannister', 'Catelyn-Stark', 'Irri']
Meme: 9 263.4489545229472 ['Vylarr', 'Wyman-Manderly', 'Thoren-Smallwood', 'Willow-Heddle', 'Tallad']
Meme: 10 261.38604244333703 ['Desmond-Grell', 'Harra', 'Brienne-of-Tarth', 'Brusco', 'Guyard-Morrigen']
```

```
temp = []
for i in range(F):
    temp.append([population[i], population_LIE[i]])
population = temp

memeplex = []
for i in range(m):
    meme = []
    for j in range(n):
        meme.append(population[i + (n * j)])
        j += n
    memeplex.append(meme)
for i in range(m):
    print("Memeplex -", i+1)
    for j in range(n):
        print(memeplex[i][j])
```

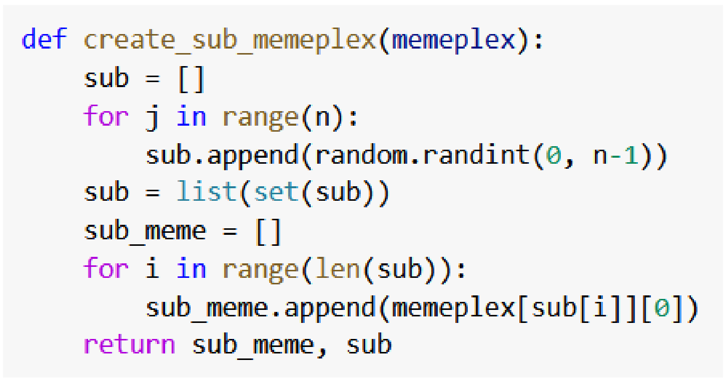
```
Memeplex - 1
[['Xaro-Xhoan-Daxos', 'Daenerys-Targaryen', 'Luwin', 'Orell', 'Jafer-Flowers'], 437.83603432746315]
[['Mellei', 'Shadrich', 'Morrec', 'Robar-Royce', 'Lark'], 256.6725110498163]
[['Bartimus', 'Arthur-Dayne', 'Ricasso', 'Nurse', 'Hagen'], 215.9376163074755]
[['Harys-Swyft', 'Narbert', 'Manfrey-Martell', 'Brusco', 'Hagen'], 177.1312390590427]
[['Mully', 'Gunthor-son-of-Gurn', 'Skinner', 'Stiv', 'Tickler'], 163.64071122148565]
[['Palla', 'Josmyn-Peckledon', 'Hot-Pie', 'Dryn', 'Eustace-Brune'], 156.71149644292584]
[['Aenys-Frey', 'Arstan', 'Areo-Hotah', 'Forley-Prester', 'Hop-Robin'], 140.54581987371733]
[['Oswell-Whent', 'Jocelyn-Swyft', 'Preston-Greenfield', 'Ottyn-Wythers', 'Roger-Ryswell'], 122.90847120513484]
[['Bloodbeard', 'Maron-Greyjoy', 'Dywen', 'Rhaegar-Targaryen', 'Tysha'], 106.71439103272272]
[['Franklyn-Fowler', 'Harrold-Hardyng', 'Rakharo', 'Pono', 'Olenna-Redwyne'], 76.58838508668795]
```



```
def sort_nodes(nodes, centrality):
    for i in range(1, len(centrality)):
        key1, key2 = centrality[i], nodes[i]
        j = i-1
        while j >= 0 and key1 > centrality[j] :
                centrality[j + 1] = centrality[j]
                nodes[j + 1] = nodes[j]
                j -= 1
        centrality[j + 1] = key1
        nodes[j + 1] = key2
    return nodes

def sort_degree_centrality(nodes):
    centrality = nx.degree_centrality(GOT.subgraph(nodes))
    cen_measure = []
    for i in nodes:
        cen_measure.append(centrality[i])
    return sort_nodes(nodes, cen_measure)
```



```python
def LDR(meme):
    # New Meme to improve the worst Meme(Frog) w.r.t. the Local/Global Best Meme (Frog)
    new_meme = []
    N1 = []
    for i in meme: # For each Memetype in the Meme
        N1.extend(list(GOT.neighbors(i)))
    N1 = list(set(N1))
    # Sort Function to sort the One Hop Neighbors of the memetype based on the Degree Centrality Metric
    SN1 = sort_degree_centrality(N1)
    for i in range(k):
        new_meme.append(SN1[i])
    return new_meme

def LDR_random():
    temp = []
    temp.append([random.choice(list(weighted_degrees)) for i in range(k)])
    return temp
```



```python
def create_sub_memeplex(memeplex):
    sub = []
    for j in range(n):
        sub.append(random.randint(0, n-1))
    sub = list(set(sub))
    sub_meme = []
    for i in range(len(sub)):
        sub_meme.append(memeplex[sub[i]][0])
    return sub_meme, sub
```

```python
for x in range(itr):
    Px = memeplex[0][0][0]
    Px_LIE = memeplex[0][0][1]
    for i in memeplex:
        for j in i:
            if j[1] > Px_LIE:
                Px = j[0]
                Px_LIE = j[1]
    for i in memeplex:
        sub_memeplex, tempz = create_sub_memeplex(i)
        Pb = sub_memeplex[0]
        Pw = sub_memeplex[-1]
        temp1 = LDR(Pb)
        temp2 = LDR(Px)
        temp = LIE(Pw)
        if LIE(temp1) > temp:
            Pw = temp1
        elif LIE(temp2) > temp:
            Pw = temp2
        else:
            Pw = LDR_random()
        sub_memeplex[-1] = Pw
        sub_prob = LIE(Pw)

        for j in range(len(tempz)):
            if i[tempz[j]][0] != sub_memeplex[j]:
                i[tempz[j]][0] = sub_memeplex[j]
                i[tempz[j]][1] = sub_prob
```

```python
population = []
population_LIE = []
for i in memeplex:
    for j in i:
        population.append(j[0])
        population_LIE.append(j[1])
sort_memes(population_LIE, population)
for i in range(len(population)):
    print(population[i], population_LIE[i])
```

```
['Robb-Stark', 'Catelyn-Stark', 'Jaime-Lannister', 'Eddard-Stark', 'Tywin-Lannister'] 459.46357192281556
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Catelyn-Stark', 'Robb-Stark', 'Tyrion-Lannister', 'Jaime-Lannister', 'Eddard-Stark'] 459.3598198013046
['Catelyn-Stark', 'Robb-Stark', 'Tyrion-Lannister', 'Jaime-Lannister', 'Eddard-Stark'] 459.3598198013046
['Catelyn-Stark', 'Robb-Stark', 'Tyrion-Lannister', 'Jaime-Lannister', 'Eddard-Stark'] 459.3598198013046
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark', 'Tyrion-Lannister'] 459.3598198013046
['Tyrion-Lannister', 'Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark'] 459.3598198013045
['Tyrion-Lannister', 'Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark'] 459.3598198013045
['Tyrion-Lannister', 'Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark'] 459.3598198013045
['Tyrion-Lannister', 'Jaime-Lannister', 'Catelyn-Stark', 'Eddard-Stark', 'Robb-Stark'] 459.3598198013045
```

FUTURE SCOPE

The project mainly focuses on designing an algorithm that identifies nodes with maximum influence, but can be expanded to include Budgets and Costs where the cost to engage with an influential node and the allotted budget can be kept in mind to design an algorithm to cater to the factors.

REFERENCES

- Jianxin Tang, Ruisheng Zhang, Ping Wang, Zhili Zhao, Li Fan, Xin Liu, A discrete shuffled frog-leaping algorithm to identify influential nodes for influence maximization in social networks, Knowledge-Based Systems, Volume 187, 2020, 104833, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2019.07.004. (https://www.sciencedirect.com/science/article/pii/S0950705119303089)
- Banerjee, S., Jenamani, M. & Pratihar, D.K. A survey on influence maximization in a social network. *Knowl Inf Syst* 62, 3417–3455 (2020). https://doi.org/10.1007/s10115-020-01461-4
- https://towardsdatascience.com/a-survey-on-shuffled-frog-leaping-algorithm-d309d0cf7503
- Maaroof, B.B., Rashid, T.A., Abdulla, J.M. *et al.* Current Studies and Applications of Shuffled Frog Leaping Algorithm: A Review. *Arch Computat Methods Eng* (2022). https://doi.org/10.1007/s11831-021-09707-2