SHANGHAI JIAOTONG UNIVERSITY

# Influence Maximization in Social Networks

Wei Pengchao,5141309020

*May 27, 2018*

Version: My First Draft

# Abstract

In recent years, due to the surge in popularity of social-networking web sites, considerable interest has arisen regarding influence maximization in social networks. Given a social network structure, the problem of influence maximization is to determine a minimum set of nodes that could maximize the spread of influences. The main purpose in influence maximization, which is motivated by the idea of viral marketing in social networks, is to find a subset of key users that maximize influence spread under a certain propagation model. In this project, I implement greedy heuristic algorithm,random heuristic algorithm and optimized greedy heuristic algorithm for influence maximization in social networks. Dataset is network of Arxiv High Energy Physics Theory category and has 9877 nodes and 51971 edges. Experiments on this real-world dataset "ca-HepTh.txt" have shown that the optimizated greedy algorithm has competitive performance relative to greedy algorithm and random heuristic algorithm in terms of propagation efficiency and influence and gives better results for running time in social networks.

# Keywords

Influence maximization; Greedy heuristic algorithm; Random heuristic algorithm; Optimized greedy algorithm; Propagation efficiency; Social network

# Contents

# 1 Introduction

In the last decade, social network analysis has drawn much attention due to its widespread applicability. A social network is a social structure made up of individuals who are tied by one or more specific types of relationship or interdependency, such as friendship, co-authorship, common interest, or financial exchange, to name a few. Nowadays, many worldwide social- networking web sites, such as Facebook and Twitter, are very popular since users can share their thoughts and comments with their friends and also bring small and disconnected social networks together. In 2011, Facebook and Twitter already had more than 600 million and about 90 million active users, respectively. Hence, marketing on online social networks shows great potential to be much more successful than traditional marketing techniques. In many enterprises, the budget of advertisement spending on worldwide social-networking sites is almost the same or even in excess of that spent in traditional ways.

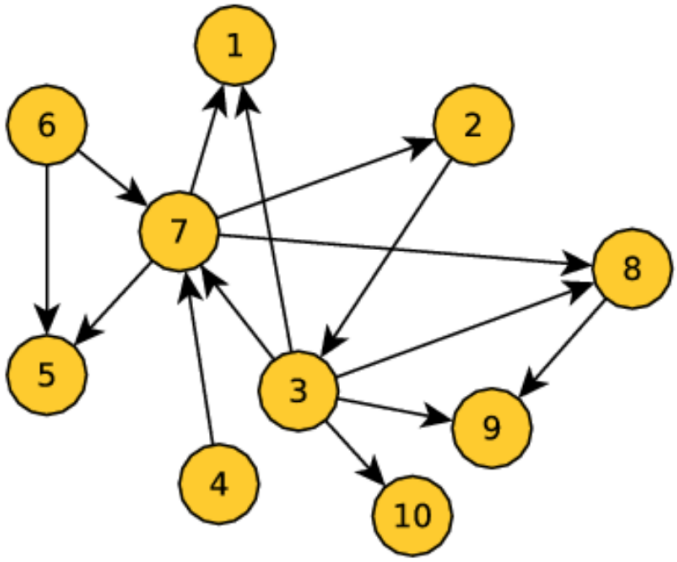


**Fig. 1.1.:** Graph Representing Social Network

A social network normally represented by a graph $G(V, E)$. Where $V$ represents the set of nodes and $E$ represents set of relationships. Figure 1.1 shows a directed social network. The set $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ is the set of nodes, and the set $E = \{(7, 1), (6, 7), (6, 5), (7, 5), (7, 2), (7, 8), (4, 7), (3, 7), (3, 8), (3, 9), (3, 1), (3, 10), (8, 9), (2, 3)\}$ is the set of edges. An undirected social network is the social network where edges are undirected all others remain same.

## 1.1 Motivation and Importance

The rapid growth and popularity of online social networking sites have brought a great deal of attention to social networks. Beside a means of communication, online social networks provide immense sources of information, experience and innovations that enable everyone from everywhere to create, exploit, or spread content through the network via internet links. More importantly, social interaction plays a central role in shaping political or social movements and debates. Social network analysis can help extracting worthwhile knowledge, controlling methods of exchanging data, maximizing acquisi- tion of information in the network and also designing improved social networks with appropriate facilities of dissemination.

A lot of studies have been done in the context of information diffusion in social networks. Precisely, information diffusion is a research domain that concerns with the processes of dissemination of information and opinion sharing among members of a social network. According to a recent survey, studies conducted in this field include three general branches as following: “Detecting Interesting Topics”, “Modeling Diffusion Processes” and “Identifying Influential Spreaders”. The latter branch is what we study in this article, which is known as “influence maximization”.

## 1.2 Problem Statement

When news and innovations arise in a social network, they usually begin to spread through the network from person to person, in a virus manner, to achieve as large individuals as possible. The extent of diffusion in the network mainly relates to the mutual relationships of its members. Consider the phone network of a group of individuals. If we send a message including a rumor to someone in this network, he will inform those in his contacts about the new message. They will either accept or ignore the rumor. In fact, if they are affected by the sender, they will believe the new announcement and begin to spread it through the rest of the network using their influence on their friends. Depending on the initial receiver of the message, final amount of receivers in the network would be different. This process is called “word-of-mouth” effect in social networks and is so much operational in commercial intentions. Recently, online social communities have become the target of many companies as a way of advertising new products. These companies aim to find the most influential individuals who are most suitable for pro- moting their brands and absorbing the most customers. They give free or discounted samples of their product to these particular persons and trigger a large cascade of further adoptions in the whole network subsequently. Therefore, proficient adoption of an exact strategy for specifying potential trendsetters is of great interest; in order to gain the most profit

utilizing their influence. These are some examples which the influence maximization problem can cope with. Given a social network graph, a seed size k, and a known influence cascade model, influence maximization is the problem of choosing a set of k influential seed nodes in the graph. Starting from these source nodes, we aim to maximize the spread of information – rumor, innovation, disease – under the influence propagation model in the network. As mentioned above, identifying the influential spreaders (seeds) has a different meaning in each kind of diffusion network. For example in the blogosphere, it means selecting a set of blogs and websites that broadcast the information in a broad range of others. In epidemiology, it consists of finding a set of persons that together are most likely to spread a virus to the greatest number of persons in the population. Identifying these individuals can help cont- rolling the disease transmission in the network. Finally, in viral marketing, the problem reduces to finding a set of trendsetters that absorb the most number of customers to adopt a product.

Formally speaking, a social network is generally modeled as an undirected graph $G(V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the vertex set and E = $\{(vi, vj)|$ there is an edge from $v_i$ to $v_j$ $\}$ is the set of edges. A node represents an individual, and an edge between two nodes represents some kind of relationship (friendship or co-authorship, etc.). A node is marked as active if it has adopted an idea or an innovation, or as inactive if it has not. Thus, the problem of the influence maximization is given below:

**(Influence Maximization Problem)** Given a social network $G = (V, E)$, the output is to determine a set of seeds (i.e., nodes) such that these seeds could spread their influence to other nodes with the purpose of maximizing the number of nodes affected by the seeds.

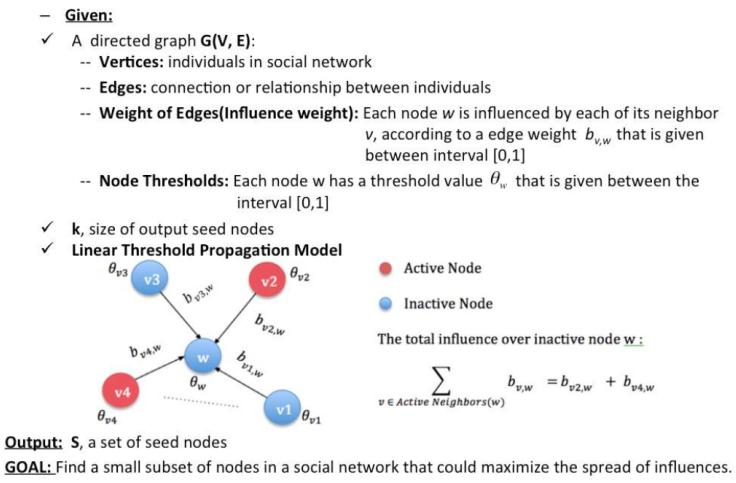


**Fig. 1.2.:** Original Problem Statement

In this project, I implement greedy heuristic algorithm,random heuristic algorithm

and optimized greedy heuristic algorithm for influence maximization in social networks. Dataset is Collaboration network of Arxiv High Energy Physics Theory category and has 9877 nodes and 51971 edges. Experiments on this real-world dataset "ca-HepTh.txt" have shown that the optimizated greedy algorithm has competitive performance relative to greedy algorithm and random heuristic algorithm in terms of propagation efficiency and influence and gives better results for running time in social networks.

# 2 Background and Related Work

With the development of the social networks, there are many researches on influence maximization in social network in recent years. The literatures[1][2] first formulate influence maximization problem as an algorithm problem. And these articles describe the basis theory of the social networks in influence maximization problems, including some basic algorithms. Kempe and other scholars put forward a series of articles to explain the definition of the influence maximization problem and prove that optimization problem is NP-hard[2]. This article also introduces two basic information diffusion model: LT (Linear Threshold model)and IC (Independent cascade model).The greedy algorithm turns out to have plenty of issues, which include the time complexity is too high. And with the network's size especially the large-scale network's size is getting bigger, the algorithm's efficiency becomes lower.

There are a lot of related researches completed under independent cascade model. the article proposed in [4] describes a CELF (Cost-effective Outbreak Detection) algorithm, the algorithm is based on the influence maximization problem and use the submodule function of the problem, the experiment in the article proves that their algorithm has a huge increase in time complexity and in performance, but the efficiency of the algorithm in large-scale network is still a drop in the bucket. Kimura and his fellow partner put forward heuristic strategy based on the shortest path in the literature [5], and design more efficient algorithm solution to influence maximization according to the strategy , efficiency of the algorithm is not very ideal in actual network because of the limitation of network size.

This article chooses LT as the research model. And there are a lot of articles proposing some algorithm based the article [3]. Chen et al. [7] study the influence maximization problem in the linear threshold model. The article first show that computing exact influence in general networks in the linear threshold model is NP-hard. Based on the fast computation in DAGs (which is proves can be done in linear time in the article), Chen propose the first scalable influence maximization algorithm (LDAG: local directed acyclic graph) tailored for the linear threshold model. The simulation shows it performs consistently among the best algorithms and it also has the stable performances on real world networks. There are a lot different algorithms as LDAG is based on the greedy algorithm [6, 7, 10]. Since the greedy algorithm is proved to be the optimal solution for the influence maximization problem, improve

the greedy algorithm still will be a research trend.

The solution for influence maximization problem is not limited to above method. Kimura et al. [8] address the problem of control the spread information transmission by minimizing the propagation of undesirable things, such as block a limited number of links in a network. Kimura also propose a method for efficiently finding a good approximate solution to this problem based on a naturally greedy strategy. Ceren Budak et al. [9] study the notion of competing campaigns in a social network and address the problem of influence limitation by using the notion of limiting campaigns to counteract the effect of misinformation. The article also study the influence limitation problem in the presence of missing data and propose a prediction algorithm that is based on generating random spanning trees which the performance of this approach turned out to be outstanding.

# 3 Algorithms and Models

In this section, I will first introduce the existing basic models of influence in the area of influence maximization. After that, I will introduce three classic algorithms of influence maximization used in this project.

## 3.1 Basic Models

Independent Cascade Model (ICM) and Linear Threshold Model (LTM) are two classic graph based approaches which exhibit the process of influence propagation in a social network. Both approaches model a social network as a directed graph in which nodes and links represent individuals and relations between them respectively. A node is active if it has adopted the content spreading through the graph or it is inactive otherwise. Each link of the graph is associated with an influence probability showing the degree of influence of the tail node on the head of the link. When a node becomes active, it can send the content to its neighbors by its outgoing links and affect them with a specific probability. In the following, we introduce these models in more details.

### 3.1.1 Independent Cascade Model(ICM)

ICM is a popular diffusion model which has been widely studied in the context of influence maximization. It describes an iterative propagation process, focusing on the sender of the information. Flow of diffusion starts from the initial seed nodes which are all active at first. Nodes can have two states, (i) Active: It means the node already influenced by the information in diffusion. (ii) Inactive: node unaware of the information or not influenced. At the beginning of ICM process, few nodes are given the information 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 neighbors. In spite of its success, the same node will never get another chance to activate the same inactive neighbor. The success depends on the propagation probability of their tie. Propagation Probability of a tie is the probability by which one can influence the other node. In reality, Propagation Probability is relation dependent, i.e., each edge will have different value. However, for the experimental purpose, it is often considered to be same for all ties. The process

continues in discrete time units, till no more nodes transfer to active state. The independent cascade model is a probabilistic model. When a node $v$ is activated, it will attempt to activate its unactivated outbound neighbor node $w$ with probability $P_{vw}$. This attempt is only performed once, and between these attempts is Independent of each other, that is, the activation of $v$ on $w$ will not be affected by other nodes.

At present, there are many researches on the maximization of influence of the independent cascaded model. The characteristic of the IC model is that it only considers the activation relationship between $u$ and the outbound neighbor $w$, and does not consider the impact of other inbound neighbors of $w$ on $w$. However, because it is a probabilistic model, its activation process is uncertain, and the final results obtained by activating the same network and the same seed nodes may vary greatly.

### 3.1.2 Linear Threshold Model(LTM)

In the case of LTM, in addition to the influence probability on each link, we need an influence threshold for each node in the network. As well as ICM, the propagation proceeds iteratively in discrete time steps while focusing on the receiver node. In fact, this model investigates the simultaneous impressions of a node's neighbors on its status. Thus, a node becomes active by its activated neighbors, if the summation of influence degrees on the incoming links exceeds its threshold amount. Again, the diffusion process ends when no transition happens to active state. In both cases of LTM and ICM, an activated node remains active and does not change its status anymore.

The linear threshold model is a value accumulation model. It has an activation threshold $\theta_v \in [0, 1]$ for each node $v$.The information propagation process of the linear threshold model is as follows:
(1) Any node $v$ in a given set is randomly assigned a threshold $\theta_v \in [0, 1]$, which indicates how difficult the node is affected. The smaller the $\theta_v$ is, the more vulnerable the node $v$ is, and the larger $\theta_v$ is, the more difficult it is to affect the node $v$. Node $v$ can be activated only when the influence of its newly active neighbor on node $v$ is greater than the threshold.
(2) $v$ is affected by its inbound neighbors $w$ with $b_{wv}$, and $b_{wv}$ satisfies:

$$\sum_{w \in in(v)} b_{wv} \leq 1$$

Where $in(v)$ is the set of inbound neighbors for $v$.

(3)Given the initial set of active nodes A (all other nodes in the network are inactive), a threshold is arbitrarily assigned to each node in the network. At time t, all active nodes remain active at time t-1. And, when the sum of the influence of the neighboring nodes of node $v$ is greater than the threshold of node $v$ at this moment, node $v$ is activated, that is, the condition that node $v$ is activated is that the cumulative effect of $v$-activated neighboring neighbors on $v$ is greater than $v$. The activation threshold is as follows:

$$\sum_{w \in in(v), active(w) \neq 0} b_{wv} \geq \theta_v$$

(4)After node v is activated, it will affect its neighbors at the next moment and repeat the above process.
In the LT propagation model, the propagation process ends when the sum of the influence of any of the active nodes already existing in the network cannot activate their neighbor nodes that are in an inactive state.

The LT model is characterized by its activation process is determined. When we activate the same graph with the same seed node, the final propagation range is exactly the same. And when a node $v$ is activated, it will try to activate each of its unactivated outbound neighbors $w$, even if this activation attempt does not enable $w$ to be activated, but $b_{vw}$ will be accumulated, and then other nodes to $w$ The activation helps, which shows that the activation process of the LT model is a cooperative activation process, and each activation attempt is accumulated.

In social network, almost each node is not independent. Linear threshold model is a good match to the herd behaviour in real life, it's conformed to the basic logic of social information transmission. In this project, we will use the linear threshold model as our research model.

## 3.2 Classic Algorithms

Many solutions have been purposed for the vertex cover problem. As it is NP-complete, there is no polynomial algorithm constructing an optimal vertex cover. Most of the methods are approximation algorithms and heuristics. In this section, I will give a rapid overview of algorithms used in this project.

### 3.2.1 Greedy Heuristic Algorithm

The main content of the greedy algorithm is as follows: first of all, choose k nodes in the network, then calculate the influence of each node above the whole set of

users in network, it divides the number of transmission, the biggest results from the set of nodes will join the initial seed after accumulation ;Choose k node in the network outside the seed collection, each of these k nodes try to join each the seed combination, then calculate the influence of each node above the whole set of users in network, accumulative average is the each node's influence divided by the number of transmission, the average maximum of the nodes is added to the seed collection; Repeat these steps until the k node seed selection as initial set of nodes completed.

**Algorithm 1 Greedy Algorithm**

**Input: G=(V,E), LT Model, seed set size k**

**Output: seed set S, the nodes' count influenced by seed set**

**1: Initial S=Φ,read graph G**
**2:for v in G,INF(v)= $|\sigma_{max}(v)|$**
**3:for i=1 to k do:**
**Random select k nodes set S from the network**
**for v in S:**
**INF(v)= $\sigma(v)T^{-1}$, T is the number of transmission**
**Select v=max{INF(v)}**
**S= SUv**
**then compute σ(v)**
**end for**
**4: return S, σ(S)**

Fig. 3.1.: Greedy Algorithm

### 3.2.2 Random Heuristic Algorithm

In this algorithm, we need to pick k nodes at random from nodes set and add it to best nodes.

### 3.2.3 Optimized Greedy Heuristic Algorithm

The main content of this algorithm is deleting some nodes from Graph $V$ to complete a new Graph in each round of greedy algorithm. The following content is about how to reasonable choosing the nodes which are about to be deleted. First, if node $v$ can influence $u$ in $(k+1)_{th}$ step, so a series of activation node from step 1 to step $k+1$ can be assumed as $\{v, w_1, w_2, ..., w_{k-1}, w_k, u\}$ in order. We can measure node $v$'s influence to active node $u$ as $INF_u(v)$. Our study is based on the LT(Linear Threshold) model, so it has serval variables including influence weight $b_{v,w}$ for link $(v, w)$ and activation threshold $\theta_v$ for node $w$.

In LT model, there can be serval nodes activating an inactive node at the same time. If node $u$ is active and $w_k$ is one of the nodes which is activating node $u$. So node $w_k$'s influence $INF_u(w_k)$ can be measured as $INF_u(w_k) = b_{wk,u}\theta_u^{-1}$. If any node $v, w$ in the Graph $V$ satisfies $b_{v,w} > \theta_w$ then set the equation $b_{v,w} = \theta_w$.

Then we can infer if we want to obtain the node $v$'s influence to $u$, we can get it from combine the influence from $v$ to $w_k$ and$w_k$ to $u$.

**Algorithm 2 Improved Greedy Algorithm:IZ**

**Input: G=(V,E), LT Model, seed set size k,float r**

**Output: seed set S, the nodes' count influenced by seed set**

**1:Initial S=Φ,read graph G**
**2:for v in G ,INF(v)=|σ(v)|T$^{-1}$**
**3:for i=1 to k do:**
**Random select k nodes set S from the network**
**for v in S:**
**INF(v)= σ(v)|T$^{-1}$**
**T is the number of transmission**
**Select v=max{INF(v)}**
**V=V-Z$_\gamma$(v)**
**S= S ∪ v**
**then compute σ(S)**
**end for**
**4: return S, | σ(S)|**

Fig. 3.2.: Optimized Greedy Algorithm

So in greedy algorithm we can delate some nodes which are the core influence zone of initial node. And the size of node set in the graph is getting lower in each round, it will reduce the time complexity of the algorithm.

# 4 Implementation Setup and Precedure

Most of the communication and social networks have power-law link distributions, containing a few nodes that have a very high degree and many with low degree [10,11]. Therefore we produce random graphs with power-law edge distribution besides other random graphs. We conduct several experiments on collaboration network of Arxiv High Energy Physics Theory category with different nodes that were selected to evaluate the performance of optimized greedy heuristic algorithms and compare it with greedy heuristic algorithm and random heuritic algorithm.

## 4.1 Environment Setup

The code is written in Python 3.6, and all the experiments are run on Mac OS X 10.10.5 machine with 2.5GHz Intel Core i5 CPU and 4GB memory. The code is available in Appendix-A. It includes the implementation of our algorithm and process of the dataset to create a random graph. The software we used in this project is Anaconda3-5.1.0.

## 4.2 Dataset

Arxiv HEP-TH (High Energy Physics - Theory) collaboration network is from the e-print arXiv and covers scientific collaborations between authors papers submitted to High Energy Physics - Theory category. If an author i co-authored a paper with author j, the graph contains a undirected edge from i to j. If the paper is co-authored by k authors this generates a completely connected (sub)graph on k nodes.

The data covers papers in the period from January 1993 to April 2003 (124 months). It begins within a few months of the inception of the arXiv, and thus represents essentially the complete history of its HEP-TH section. This dataset is a directed graph and contains 9877 nodes and 51971 edges.

Tab. 4.1.: Dataset Statistics

| Dataset statistics | |
|---|---|
| Nodes | 9877 |
| Edges | 25998 |
| Nodes in largest WCC | 8638(0.875) |
| Edges in largest WCC | 24827(0.955) |
| Nodes in largest SCC | 8638(0.875) |
| Edges in largest SCC | 24827(0.955) |
| Average clustering coefficient | 0.4714 |
| Number of triangles | 28339 |
| Fraction of closed triangles | 0.1168 |
| Diameter(longest shortest path) | 17 |
| 90 percentile effective diameter | 7.4 |

## 4.3 Implementation Precedure

Step 1: Read graph file from dataset ca-HepTh.txt and create a random graph;
Step 2: Set threshold=10 and the value of totalnodes;
step 3: Get influence map

```
find the influence of each vertex
create a dict vert_influence with key as node and value as the set of nodes influenced by node
best_node = node with the largest influence
best_set = set(best_node)
uninfluenced_set = nodes_set

repeat k times
  uninfluenced_set = uninfluenced_set - influence set of best node
  for every node in uninfluenced set
    num_new_influences = len(influence set of node from the dict created earlier    intersection    uninfluenced set)
  find best_node - the node which produces maximum num_new_influences
  add best_node to best_set
return best_set
```

Fig. 4.1.: Get Influence Map

step 4: Remove any nodes that are there in source nodes from influenced_nodes.
step 5: Calculate the influence and size of influenced set of each algorithm.
step 6: Draw the influence-curve and execution time curve of each algorithm.

Tab. 4.2.: Varialbe Definition

| Varialbe | Definition |
|---|---|
| graph_snaps | graph snapshots |
| nodes_set | set of nodes in the complete graph |
| k | number of nodes to influence initially |
| step_size | number of nodes to add to the opt set every iteration |

# 5 Experimental results and Discussion

In this section, I will present the experiment results of random heuristic algorithm, greedy heuristic algorithm and optimized heuristic algorithm. You can get the set of nodes that maximize the influence of these three algorithms in the result.txt file.

## 5.1 Influence Performance

The following tabel and figures describe the size of influenced set of each algorithm.

**Tab. 5.1.:** Size of Influenced Set of Each Algorithm with Different Total Nodes

| Toal Node | 20 | 50 | 100 | 150 |
|---|---|---|---|---|
| Random Heuristic Algorithm | 3892.9 | 3925.2 | 3950.5 | 3936.2 |
| Greedy Heuristic Algorithm | 3880 | 3874 | 3858 | 3791 |
| Optimized Greedy Heuristic Algorithm | 4061 | 4255 | 4519 | 4657 |

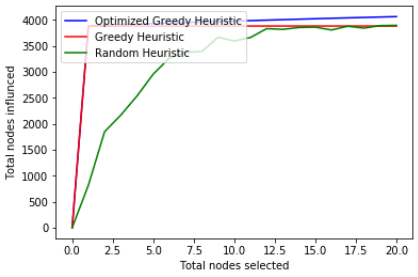


**Fig. 5.1.:** TotalNode=20

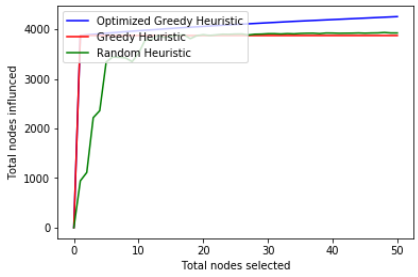


**Fig. 5.2.:** TotalNode=50

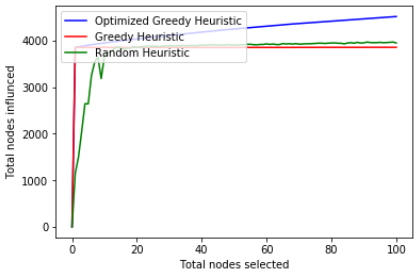


**Fig. 5.3.:** TotalNode=100

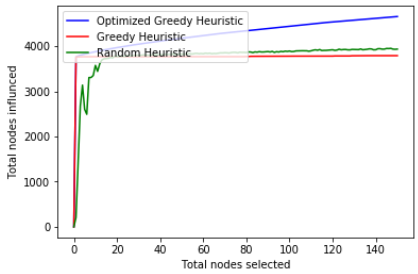


**Fig. 5.4.:** TotalNode=150

Through these four figures we can see that optimized greedy algorithm performs slightly better than the original greedy algorithm and outperforms heuristic algorithms.

## 5.2 Running Time Performance

We compared the running time performance of optimized greedy heuristic algorithm to running time performance of original greedy heuristic and random heuristic algorithm. We experimented these algorithms on the dataset mentioned in the previous section. It is obvious that optimized greedy heuristic algorithm performs better than original heuristic algorithm. Moreover the difference between the running time performances of optimized greedy algorithms and original heuristic algorithm become larger as the number of total node is increasing. The running time comparison of optimized greedy algorithm and other two algorithms is shown in Figures below.

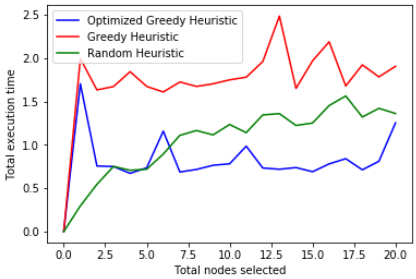


**Fig. 5.5.:** TotalNode=20

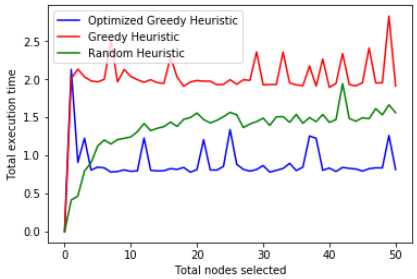


**Fig. 5.6.:** TotalNode=50

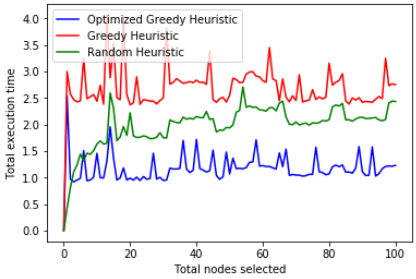


**Fig. 5.7.:** TotalNode=100

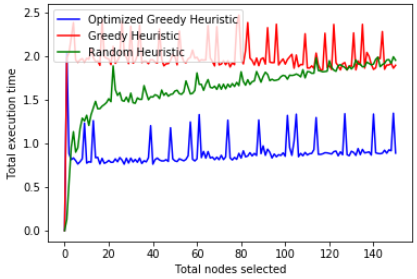


**Fig. 5.8.:** TotalNode=150

# 6 Conclusion

In this project, I worked on the influence maximization problem in social networks.First, I introduce the basic concepts of influence maximization and briefly described what Independent Cascade Model of Information Diffusion is and provided an implementation, in details, of greedy algorithm, random heuristic algorithm and optimized greedy heuristic algorithm. Coding is done in Python.Method described here can directly be used for any directed social networks. With simple modification, it can also be generalized to work with undirected social networks as well.Then I do some experiments on social networks ca-HepTh.txt and this experiments demonstrate the effectiveness and the efficiency of classic algorithms and optimized greedy algorithm. I established a series of experiment by Python language in heuristic algorithm and the improved greedy algorithm under LT model. In fact the network which is a package of python make us work easier. The heuristic algorithms we used in the experiments is the random heuristic algorithm, greedy heuristic algorithm and optimized greedy heuristic algorithm.

These experiments show that optimized greedy algorithm gives better results for running time in social networks and have a better influence performance relative to other heuristic algorithms. Next, I want to study how to change the algorithm so it can have a good performance on different social networks. And the efficiency of the algorithm is also need to be improved.

# 7 Reference

[1] P. Domingos and M. Richardson. Mining the network value of customers. In Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining,pages 57–66, 2001.
[2] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In Proceedings of the 9th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 137–146, 2003.
[3] M. Richardson and P. Domingos. Mining knowledge- sharing sites for viral marketing. In Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 61–70, 2002.
[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos,J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 420–429, 2007. Algorithm.
[5] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, pages 259–271, 2006.
[6] Wei Chen, Yajun Wang, Siyu Yang .Efficient Influence Maximization in Social Networks. KDD '09 Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining Pages 199-208,2009.
[7]W. Chen, Y. Yuan, L. Zhang, Scalable influence maximization in social networks under the linear threshold model, Proceedings of 10th International Conference on Data Mining, 88-97,2010.
[8] M Kimura, K Saito, H Motoda, Minimizing the Spread of Contamination by Blocking Links in a Network, Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence,1175-1180,2009.
[9] Ceren Budak, Divyakant Agrawal,Amr El Abbadi, Limiting the spread of misinformation in social networks, WWW '11 Proceedings of the 20th international conference on World wide web, 665-674,2011.
[10]A Goyal,W Lu,LVS Lakshmanan, Simpath: An efficient algorithm for influence maximization under the linear threshold model, IEEE 11th International Conference on Data Mining (ICDM), 211 – 220,2011.
[11]Erdos, P. and Renyi, A., On random graphs, Publicationes Mathematicae 6, 290–297 (1959) [14]Goh K-I, Kahng B, Kim D: Universal behaviour of load distribution in scale-free networks. Phys Rev Lett 87(27):278701, 2001

# List of Figures

# List of Tables

# Aappendix-code

A.

This appendix is code of my project. Code is on the followinging pages.

```
#python3.6

import sys
import random
from Graph import *
from ReadDataset import *
from Heuristic import *
from InfluenceUtility import *
sys.setrecursionlimit(100000)
import matplotlib.pyplot as pyplot
import time


def main () :
    edges_list, weight_list, nodes_set = ReadGraphFile ()
    graph_snaps = CreateRandomGraphs (50, edges_list, weight_list)
    step_size = 1;
    threshold = 10;
    totalNodes = range(0, 21);

    optimizedGreedyHeuristic = [];
    greedyHeuristic = [];
    randomHeuristic = [];
    optimizedGreedyHeuristic.append(0);
    greedyHeuristic.append(0);
    randomHeuristic.append(0);
    influenceMap = getInfluenceMap(nodes_set,graph_snaps,threshold);
    optimizedGreedyHeuristicSelectedSet = set();
    greedyHeuristicSelectedSet = set();
    randomHeuristicTime = [];
    randomHeuristicTime.append(0);
    optimizedGreedyHeuristicTime = [];

    optimizedGreedyHeuristicTime.append(0);
    greedyHeuristicTime = [];
    greedyHeuristicTime.append(0);
    fout=open("result.txt",'w')

    for k in totalNodes :
        print ("k = ", k)
        fout.write("k="+str(k)+'\n')

        if k == 0:
            continue;

        startTime = time.time();
        randomHeuristicCount = random_heuristic(graph_snaps, nodes_set, k, step_size,
threshold);
        randomHeuristicTime.append(time.time() - startTime);
        randomHeuristic.append(randomHeuristicCount);

        startTime = time.time();
        influenceSet = heuristic1(graph_snaps, nodes_set, k, step_size,
threshold,influenceMap,optimizedGreedyHeuristicSelectedSet);
        optimizedGreedyHeuristicTime.append(time.time() - startTime);
        print("Size of influenced set is ", len(influenceSet));
        fout.write("Size of influenced set is "+str(len(influenceSet))+'\n')
        optimizedGreedyHeuristicCount = len(influenceSet);
        startTime = time.time();
        greedyHeuristicCount =
len(heuristic2(graph_snaps,nodes_set,k,step_size,threshold,influenceMap,greedyHeuristicSel
ectedSet));
        greedyHeuristicTime.append(time.time() - startTime);
        print("Size of influenced set by greedy heuristic is ", greedyHeuristicCount);
        fout.write("Size of influenced set by greedy heuristic is "+ str(
greedyHeuristicCount)+'\n')


        print ("Size of influenced set by random heuristic is ", randomHeuristicCount)
```

```
#python3.6

class Graph :
    def __init__ (self) :
        self.edges = {}

    def add_edge (self, from_, to) :
        if from_ not in self.edges :
            self.edges[from_] = [to]
        else :
            self.edges[from_].append(to)

        if to not in self.edges :
            self.edges[to] = []

    def find_reachable_nodes (self, source_nodes) :
        # source nodes is a list of nodes
        reached = set([])
        for node in source_nodes :
            if node in self.edges :
                self.dfs (node, reached)

        return reached

    def dfs (self, node, reached) :
        for nbr in self.edges[node] :
            if nbr not in reached :
                reached.add (nbr)
                self.dfs (nbr, reached)

    def print_graph (self) :
        for node in self.edges :
            print (node, ": ", self.edges[node])

def test () :
    G = Graph ()
    G.add_edge (0,1)
    G.add_edge (1,2)
    G.add_edge (2,3)
    G.add_edge (1,4)
    print (G.find_reachable_nodes ([1]))

if __name__ == "__main__" :
    test ()
```

```
#python3.6

# Reads a text file and creates a graph as a list of edges, each edge has a random weight.

from Graph import *
import sys, random

def ReadGraphFile () :
    f = open ("ca-HepTh.txt")
    lines = f.readlines()

    edges_list = []
    weight_list = []
    nodes_set = set([])
    for line in lines :
        if line[0] == "#" :
            continue

        node1, node2 = map(int, line[:-1].split())
        edges_list.append ([node1, node2])

        # TODO: assign num parallel edges here and use it to assign a probability for each
edge.
        weight_list.append (random.random()/3)
        # weight_list.append (1)
        nodes_set.add(node1)
        nodes_set.add(node2)

    return edges_list, weight_list, nodes_set

def CreateRandomGraphs (num_graphs, edges, probs) :
    graph_snapshots = []
    for i in range (num_graphs) :
        tmp_graph = Graph ()
        for edge, prob in zip(edges, probs) :
            rand = random.random()
            if rand < prob :
                tmp_graph.add_edge (edge[0], edge[1])
        graph_snapshots.append(tmp_graph)

    return graph_snapshots
```

```
#python3.6

def find_influence (source_nodes, graph_snapshots, threshold) :
    influenced_node_count = {} # this contains nodes and count for each node
    for G in graph_snapshots :
        S = G.find_reachable_nodes (source_nodes)
        # Update influenced_node_count looking at S
        for node in S :
            if node in influenced_node_count :
                influenced_node_count[node]+=1
            else :
                influenced_node_count[node] = 1

    influenced_nodes = set([])
    for node in influenced_node_count :
        if influenced_node_count [node] > threshold:
            influenced_nodes.add (node)

    # We do not want source nodes in the influenced set. Remove any nodes that are there
in source nodes from influenced_nodes.
    for node in source_nodes :
        if node in influenced_nodes :
            influenced_nodes.remove (node)

    return influenced_nodes
```

```
#python3.6
import  InfluenceUtility as influence
import pickle
import random

def heuristic1 (graph_snaps, nodes_set, k, step_size, threshold,influenceMap,selectedSet)
:
    load_influence_map_from_file = 0
    # graph_snaps: graph snapshots
    # nodes_set: set of nodes in the complete graph
    # k: number of nodes to influence initially
    # step_size: number of nodes to add to the opt set every iteration
    uninfluencedNodes = nodes_set;
    bestNodes = set();
    maxLength = 0;
    maxNode = -1;
    for node in selectedSet:
        bestNodes = set.union(influenceMap[node],bestNodes);
        bestNodes.add(node);
        uninfluencedNodes.discard(node);
        uninfluencedNodes = uninfluencedNodes.difference(bestNodes);
    if not load_influence_map_from_file :
        f = open ("influenceMapObject.pickle", "wb")
        pickle.dump (influenceMap, f)
        f.close ()
    else :
        f = open ("influenceMapObject.pickle", "rb")
        influenceMap = pickle.load (f)
    for uninfluenced_node in uninfluencedNodes:
        new_nodes_influenced = len(set.intersection(influenceMap[uninfluenced_node],
uninfluencedNodes));
        if maxLength < new_nodes_influenced:
            maxLength = new_nodes_influenced;
            maxNode = uninfluenced_node;
        # print ("best nodes before", len(bestNodes))
    bestNodes.add(maxNode);
    bestNodes = set.union(bestNodes,influenceMap[maxNode]);
    selectedSet.add(maxNode);
    # print ("uninfluenced nodes", len(uninfluencedNodes))
    return  bestNodes;

def heuristic2 (graph_snaps, nodes_set, k, step_size, threshold,influenceMap,selectedSet)
:
    load_influence_map_from_file = 0
    # graph_snaps: graph snapshots
    # nodes_set: set of nodes in the complete graph
    # k: number of nodes to influence initially
    # step_size: number of nodes to add to the opt set every iteration
    uninfluencedNodes = nodes_set;

    bestNodes = set();
    maxLength = 0;
    maxNode = -1;
    for node in selectedSet:
        bestNodes = set.union(influenceMap[node], bestNodes);
        bestNodes.add(node);
        uninfluencedNodes.discard(node);

    if not load_influence_map_from_file:
        f = open("influenceMapObject.pickle", "wb")
        pickle.dump(influenceMap, f)
        f.close()
    else:
        f = open("influenceMapObject.pickle", "rb")
        influenceMap = pickle.load(f)

    for uninfluenced_node in uninfluencedNodes:

        new_nodes_influenced = len(set.intersection(influenceMap[uninfluenced_node],
```