Spring 5-22-2017

Influence Detection And Spread Estimation in Social Networks

Madhura Kaple
San Jose State University

Influence Detection And Spread Estimation in Social Networks

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Madhura Kaple

May 2017

The Designated Project Committee Approves the Project Titled

Influence Detection And Spread Estimation in Social Networks

by

Madhura Kaple

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2017

ABSTRACT

Influence Detection And Spread Estimation in Social Networks

by Madhura Kaple

A social network is an online platform, where people communicate and share information with each other. Popular social network features, which make them different from traditional communication platforms, are: following a user, re-tweeting a post, liking and commenting on a post etc. Many companies use various social networking platforms extensively as a medium for marketing their products. A fixed amount of budget is alloted by the companies to maximize the positive influence of their product. Every social network consists of a set of users (people) with connections between them. Each user has the potential to extend its influence across this network. The amount of influence propagated by some users is larger as compared to others. Companies, given a fixed budget, target this subset of users to attain maximum influence spread. We can model this as an influence maximization problem. This subset of users then influence the behavior or choices of other users by methods like word of mouth, actions etc. through an influence propagation across the network. The aim of this project is to compare different known and new proposed algorithms for the influence maximization problem. We measure the efficiency of each algorithm based on the number of vertices influenced by the initial set of influencers. In addition, a comparison of computation time required for each algorithm is done using various synthetic random graphs and real world social network datasets.

ACKNOWLEDGMENTS

I take this opportunity to express my sincere gratitude to my advisor, Prof. Katerina Potika for advising me on this project. Her timely guidance and support helped me research about the topic and proceed in the right direction towards completion of the project. I would like to thank her for her time and efforts for advising me on this project.

I am grateful to Prof. Sami Khuri and Prof. James Casaletto for being a part of my project committee. I thank them for their valuable comments and appreciate their time and efforts. I also would like to thank my family and friends for their continuous support and best wishes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Today, social networks have become a popular communication media among people. Facebook, Twitter, Linkedin are some of the examples of popular social networking platforms. A social network is modeled as a graph with people acting as vertices and relationships between people form the edges of the graph. These networks are useful means to spread information among people. Humans form their opinions based on the opinions of their friends and peers. They adopt a behavior based on the behavior of others. Also, their decisions get influenced by their peers and friends. Viral marketing is a process where companies use social network platforms for the marketing of their products. It is observed that some people are acting as influencers, meaning they are more capable of influencing bigger groups of people than others related to a topic, opinion, behavior or product. These people are termed as *seed set* of *influencers*. A person is said to be influenced by the *seed set* of *influencers* if it is convinced to adopt an opinion, behavior or product. Companies having limited budget for marketing, try to activate or recruit these *influencers* in the network to achieve maximum popularity of their product. Identifying such a set of *k-influencers* is termed as the *influence maximization problem*.

The influence maximization problem was formulated as a discrete optimization problem by the authors Kempe et al., in paper [11]. The paper defines that a vertex is activated if it adopts an idea, product or behavior from *influencers* in the *seed set*. Also, once activated, a vertex remains in active state. It is stated as, a social graph contains people as vertices and the edges correspond to the relationship between people. Also, the potential influence between the users corresponds to edge weights. Given a budget *k*, find a *seed set* of *k-influencers* such that on the activation the influence spread is maximized across social graph. This influence spread occurs in discrete steps. The influence spread of the *seed set* is measured using various propagation models. Linear Threshold and Information Cascade are two popular information propagation models. Identifying influencers and measuring the spread of influence can help in making recommendations to the users in a network [5].

1.1 Problem Formulation

Consider a social graph, G = (V,E) and a fixed budget *k*. The *influence maximization problem* is to find an initial set of *k-influencers* called *seed set* $S_0 \subset$ V where, $\mid S_0 \mid <=$ k such that influence spread of S_0 under given diffusion model is maximized [22]. This optimization problem is NP-hard. A greedy approximation algorithm is used as a solution to this problem. This greedy algorithm guarantees an efficiency of at least 63% approximation. It works greedily by selecting one vertex at a time and measuring the number of vertices influenced by it. This process is termed as spread estimation. The greedy approximation solution has a large number of spread estimation calls which makes it slow. Thus, it is time consuming for large graphs and depends heavily on the way we calculate the spread.

This project has the following objectives:

1. Detection of *seed set* of *k - influencers* such that it maximizes information spread across the network, using new proposed methods of *seed set* detection.

2. Measuring the spread of influence or information diffused by the *seed set* of *k-influencers*, under different scenarios in directed and undirected graphs.

3. Comparison of the new proposed algorithms against the existing greedy approximation algorithm in terms of the influence spread attained by the *k - influencer seed*

set and the computation times.

For detection of *seed set* of *k* - *influencers*, in one of our methods, we use an acyclic spanning graph based approach for undirected social network graphs. This algorithm gives faster *seed set* detection and spread estimation as compared to the greedy approximation solution. For directed graphs, the *seed set* is detected using the CELF algorithm [13]. The performance of CELF algorithm is improved by modifying it and adding the discovery of vertex cover as a preliminary step in the algorithm. The Linear Threshold model is used to compute the influence spread of the detected *seed set*. A comparison of the results obtained from the the new proposed algorithms and the existing greedy approximation solution is done.

In chapter 2, we explain the terminology and graph concepts, used in the *influence maximization* algorithms.

CHAPTER 2

Terminology

2.1 Graph Terminology

1. Graph

A Graph G = (V, E) contains a set of vertices n = | V | and edges m = | E |. The vertices are connected to each other by set of links called as edges. The graphs can be undirected as well as directed. In directed graphs, the edges connecting the vertices have directions. Graphs are used to model relationships among a collection of items.

2. Degree

An in-degree of a vertex in a graph is defined as the count of incoming edges for a vertex in the graph. Similarly, out-degree of a vertex is the count of outgoing edges from that vertex in a graph. In-degree of a vertex v is represented as $d^-(v)$. Similarly, out-degree of a vertex v is represented as $d^+(v)$

3. Representations

A graph G = (V, E) having n vertices and m edges where n = | V | and m = | E |, is usually represented using two data structures namely an adjacency matrix and an adjacency list. In an adjacency matrix representation, we use a 2D array of size nxn such that if an edge is present in between two vertices, matrix entry will be non-zero else it will be 0. In an adjacency list, for each vertex v $\in$ V we maintain a list of vertices adjacent to v. With an adjacency list, we can traverse the list and find all the vertices connected to a particular vertex by corresponding entry.

4. Breadth First Traversal (BFS)

BFS is a traversal algorithm in which we traverse the graph level wise starting from the root or starting vertex. We then visit its neighbors before proceeding to their neighbors. BFS is given in Algorithm 1.

Algorithm 1 Breadth First Search (BFS)

```
1: procedure BFS($G, s$)
2: Let Q be queue
3: Let visited be array to maintain the visited vertices
4: Initialization:  Insert s in queue Q and mark s as visited
5:    while  Q is not empty do
6:       v $\leftarrow Q.dequeue()$
7:       for  all neighbors n of v in Graph G  do
8:          if  n is not visited then
9:             Insert n in Q
10:            Mark n as visited
```

Example: Consider the graph with 12 vertices and 16 edges given in Figure 1. If we consider 0 as the starting vertex, then the BFS will be

BFS : 0, 5, 3, 2, 1, 4, 7, 8, 6, 10, 9, 11

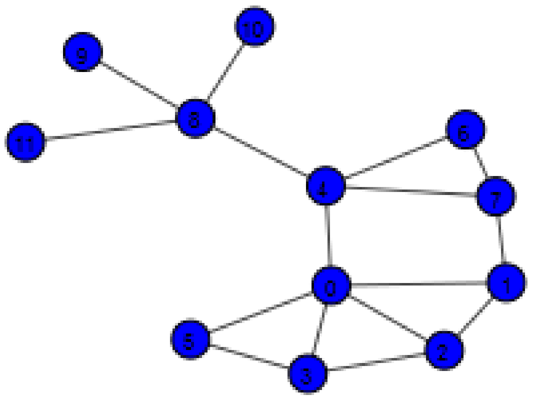

Figure 1: Example: Undirected graph

5. Vertex Cover

A vertex cover of a graph is a subset of vertices where each edge of a graph is incident to at least one vertex in the set. A minimum vertex cover is the smallest possible size of vertex cover set. The vertex cover for the graph in Figure 1 is (0, 1, 2, 3, 4, 7, 8). This is not necessarily the minimum vertex cover.

6. Acyclic Spanning Graph

Acyclic spanning graph is constructed by removing the cycles from the original graph. It has all the vertices in the original graph. The graph in Figure 1 is an undirected graph with cycles. The graph in Figure 2 represents the spanning graph constructed from the graph in Figure 1.

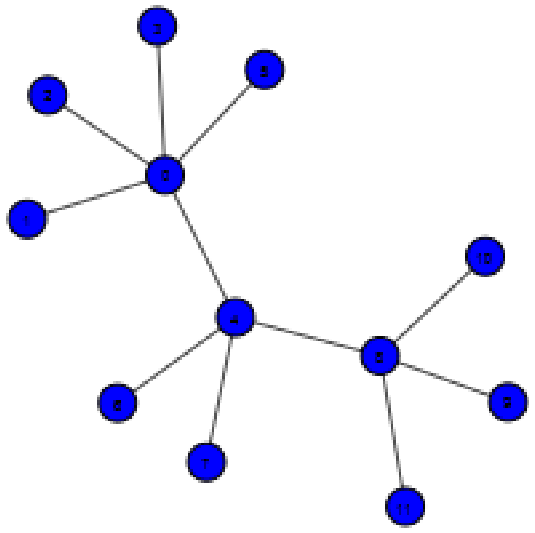

Figure 2: Example: Spanning Graph

2.2 Types of Centralities

In a social network, the centrality of a vertex defines its importance in the network. Below are some of the types of centralities in the social network graphs [4].

1. Degree Centrality

Degree centrality is determined by the count of vertices to which a vertex is connected to. In a directed graph, we define in-degree and out-degree centrality for a vertex. In-degree centrality is the count of incoming edges of a vertex whereas the number of outgoing edges of a vertex defines the out-degree centrality. Normalized degree centrality is obtained by dividing the degree centrality of a vertex by (n - 1) where n is the number of edges.

2. Eigenvector Centrality

The drawback of degree centrality is that it assumes that the vertex with the maximum number of connections is the central vertex. However, this is not true in all cases in the real world. To overcome this drawback, Eigen vector centrality accounts for the importance of the vertices that a vertex is connected to. It is given by the below formula,

$$C_e(v_i) = \frac{1}{\lambda} * \sum_{j=1}^{n} A_{j,i} C_e(v_j) \tag{1}$$

where,

$C_e(v_i)$ is the Eigenvector centrality of vertex v_i

λ is a fixed constant

A is the adjacency matrix for social graph

3. Katz Centrality

The drawback of Eigenvector centrality is that in a graph, centrality does not propagate if the vertex does not have any outgoing edges. In the case of directed graphs, where the vertex has many incoming edges but there is no outgoing edge then the centrality is not propagated from that vertex. To overcome this drawback, Katz centrality adds a bias while calculating the centrality. Due to this even if the vertex does not have any outgoing edge the centrality does not become 0. It is given as below.

$$C_{Katz}(v_i) = \alpha * \sum_{j=1}^{n} A_{j,i} C_{Katz}(v_j) + \beta \tag{2}$$

where,

$C_{Katz}(v_i)$ is the Katz centrality for vertex v_i

α is the fixed constant

β is a bias term added to avoid zero centrality

A is the adjacency matrix representation for social graph

4. PageRank Centrality

The drawback of Katz centrality is that it gives equal centrality to all its outgoing edges. However, in real world cases this is not true since all the vertices connected to the important vertices are not central. Thus, PageRank centrality solves this problem by dividing the centrality value by the out degree $d^+(v)$ of the vertex.

$$C_p(v_i) = \alpha * \sum_{j=1}^{n} A_{j,i} \frac{C_p(v_j)}{d^+(v)} + \beta \tag{3}$$

where,

$C_p(v_i)$ is the PageRank centrality for vertex v_i

α is the fixed constant

β is a bias term added to avoid zero centrality

5. Betweeness centrality

There are two types of betweeness centralities namely vertex betweeness centrality and edge betweeness centrality. Vertex betweeness centrality is defined as the number of shortest paths which pass through the given vertex divided by the number of the total number of shortest paths in a given graph. Similarly, edge betweeness centrality of an edge e is the sum of the fraction of all-pairs shortest paths that pass through the edge e.

6. Closeness centrality

It is defined by the distance of a given vertex from the other vertices in the network. The vertex which has the smallest average distance from other vertices in the network is defined to be the most central vertex.

7. Group centrality

All the centralities defined above can be defined for a group of n vertices instead of a single vertex. These are known as group centralities.

2.3 Degree Discount Heuristics

Degree Discount Heuristic is one of the techniques used for finding the set of influencers in social network. Consider a vertex v in a social graph. Let vertex u be its neighbor which is in the active state. For vertex v when computing its degree to determine if it is in the active state we do not consider the edge uv. Hence, we discount its degree. We follow similar process for all the vertices. Degree discount heuristics can be used with all cascade models.

2.4 Linear Threshold diffusion model

The Linear Threshold model is one of the popular information diffusion models. Given a social network graph with a set of vertices and edges connecting the vertices. This is a weighted graph where each edge has a weight w. A vertex can be in an active or inactive state. Once activated, the vertex will remain in the active state. Each vertex randomly selects a threshold $\theta :$ V [0, 1]. A vertex gets activated if the sum of the incoming edge weights of its active neighbors is greater than the threshold value [22]. The diffusion ends when no more vertices become active.

$$\sum_{u \in N_v} W_{u,v} >= \theta_v \tag{4}$$

Initially a set of active vertices is given known as seed set. In discrete steps we evaluate the sum of incoming edge weights of active neighbors of each vertex in the graph. If the threshold value of a vertex is smaller than the incoming sum, it becomes active and is added to the influencers set. This process ends when no more vertices are activated.

2.5 Independent Cascade Model

The Independent Cascade is another popular information diffusion model. Given a social network graph, such that the edges connecting the vertices are associated with a probability. A vertex can be in active or inactive state. Once a vertex is activated, it will not become inactive. Initially we are given a set of active vertices known as seed set. In the subsequent discrete steps, we compute the influence spread of the seed set. At each step we determine if a vertex is active by considering the probability of its edges with the active neighbors. If the probability is greater than the threshold, the vertex becomes active. However, each vertex has one chance to activate its neighbors. This process stops when no more vertices become active.

The *influence maximization* problem is a popular social network research topic. The chapter 3 gives an overview of the existing research work done to solve the *influence maximization* problem.

CHAPTER 3

Related Work

3.1 Existing methods for Influence Maximization

Influence detection and spread estimation is a popular research topic in social networks. In paper [11], authors Kempe et al. describe an approximation greedy algorithm for finding the *K-influencer seed set* in social networks. The greedy algorithm guarantees at least 63% of accuracy. Popular diffusion models like Linear Threshold and Independent Cascade models are discussed in the paper [11]. These models provide a better performance as compared to the traditional heuristics of degree and distance measures. In Linear Threshold model, each vertex is associated with a weighted threshold and in order to get activated it requires that the sum of its active neighbors must be at least greater than or equal to its threshold. Independent Cascade model is a probability based model where each vertex gets only one chance to activate its neighbors. These models are based on the sub modular functions and a greedy algorithm is used to attain the performance guarantees. The Physics publication co-authorship network is used in this paper for the experiments.

The greedy algorithm is time consuming for large graphs. As an improvisation to the greedy algorithm, CELF algorithm was proposed in the paper [13]. The paper [13] describes CELF algorithm to improvise the performance of greedy approximation solution. It uses the property of submodularity and lazy evaluations for computing the influence spread. As compared to the simple greedy algorithm, CELF algorithm performs 700 times faster. The algorithm also handles cases like vertices with different costs in a network. The algorithm is evaluated against the real world networks. The paper [13] considers the cases of unit costs as well as non-constant costs. In the equal cost, the greedy algorithm iteratively, in step k adds the vertex s_k with maximum marginal gain.

$$s_k = argmax_{s \in \frac{V}{A_{k-1}}} R(A_{k-1} \cup \{s\} - R(A_{k-1}) \tag{5}$$

where,

s_k = seed set

R is a submodular function

In the case of non-constant costs, the greedy algorithm adds the vertex with maximum benefit by cost ratio. The algorithm is terminated when no more vertices can be added to the current set.

$$s_k = \frac{argmax_{s \in \frac{V}{A_{k-1}}} R(A_{k-1} \cup \{s\} - R(A_{k-1})}{c(s)} \tag{6}$$

s_k = seed set

R is a submodular function

c(s) = cost function

However, this can give poor efficiency. Hence, for a given graph using the cost effective forward selection property, we compute the solution using both the approaches (constant weights and non-constant weights) and return the better result. It is optimal as at least one of the approaches will be close to optimal and the better result is returned. It roughly guarantees an efficiency of 31%. The algorithm explores lazy evaluation of marginal increments to reduce the computations and attain better efficiency. The Blog dataset and water distribution network are used for experiments in the paper [13].

As an extension to the CELF algorithm, the paper [9] discusses the CELF++ algorithm which works on the principle that if the previous best vertex is selected as an influencer in the current iteration, then we do not recompute the marginal gain.

For each vertex, we associate a tuple consisting of the marginal gain, flag and previous best vertex. The experiments in the paper [9] are performed using academic collaboration networks of NetHEPT and NetPHY.

The paper [8] describes an efficient algorithm SIMPATH for maximizing the influence spread under Linear Threshold model. This algorithm has faster running time due to the use of vertex cover and Lazy Forward selection optimizations. These are used to reduce the spread estimation calls. The preliminary step of the algorithm, is th discovery of vertex cover. For every vertex u ∈ C, its spread is computed on required subgraphs needed. The spread of non-vertex cover vertices is computed using Linear Threshold propagation model. Lazy optimizations are implemented using the priority queue based on the decreasing order of the marginal gains. The paper uses NetHEPT collaboration network, Last.fm: a popular music platform dataset, Flixter: a movie social platform and DBLP datasets to perform the experiments.

The paper [6] proposes another approach to find the solution for influence maximization problem. This solution is adapted from the Independent Cascade Model and extracts an acyclic spanning graph from the given social graph. The cycles in a social network will cause a feedback of influence to itself. A vertex in a social network once activated does not become inactive. Hence, by removing the cycles in a social graph we get a better influence spread model. Closeness centrality is used to construct the acyclic spanning graph. This paper describes two algorithms namely SCG-algorithm for a connected graph and SDG-algorithm for a digraph to identify the k influencer seed set. The complexity of these algorithms is $O(mn)$. The Independent Cascade Model is used to evaluate the spread of the seed set. Results show that the we obtain a better influence spread using acyclic spanning graph model. The two social network datasets used for experiments in this paper are com-Amazon communities and Enron email.

A modification of the greedy approximation solution is proposed in the paper [3]. The new greedy algorithm proposed in this paper is an improvement over the traditional greedy algorithm. It works by constructing random graphs at each step. In every step the new greedy algorithm has to traverse the entire graph whereas the traditional greedy algorithm traverses only few vertices. Hence, the paper [3] describes a mixed algorithm, where the first round uses the new greedy IC algorithm and for further rounds we use the traditional greedy algorithm. An improvement for weighted cascade model is also discussed in the paper. For this approach we use the Cohen's randomization algorithm. It is stated as given a graph, the first step is to traverse the graph once, compute all strongly connected components into one vertex with the weight being the size of the strongly connected component. For weighted cascade model too we have a mixed algorithm similar to the Independent Cascade model. The above mentioned algorithm improves the performance of traditional greedy algorithm, however, we cannot use this algorithm efficiently over large graphs. As a solution to this problem, degree discount heuristics approach is discussed in the paper. For a vertex v if u is its neighbor and u is present in the seed set, then when selecting the vertex v in the seed set we will not consider the edge uv. This principle is known as degree discount heuristics. The collaboration networks, NetHEPT and NetPHY are used for experiments in the paper.

Identification of influencers in a social network has many applications. One of the applications is use of influencers to make recommendations. The paper [5] describes a two-phase recommendation algorithm for social networks. In the first phase of the algorithm we detect the influencers in the network. This influence detection is done only in one cycle unlike Information Cascade algorithm and hence is faster. Once influencers are detected, each influencer maintains its own list of preferences of items. Thus, each influencer presents an inactive vertex with a list of its preferences based on which recommendations are made. As opposed to the traditional recommender systems, the proposed framework makes the recommendations for multiple items at once, without the explicit ratings of the items.

For the first phase of detecting influencers, a greedy algorithm to detect minimum dominating set is proposed. Once the influencers are detected these are then used in the recommendation step. For recommendation, the influencers can recommend either a single item or a list of items. In the case of multiple item recommendations, voting system will be used to determine the conflict between recommendations provided by influencers.

$$R(v, t_j) = \frac{\sum\limits_{v_i \in N(v) \cap D} sim(v, v_i).R(v, t_j)}{\sum\limits_{v_i \in N(v) \cap D} sim(v, v_i)} \tag{7}$$

where,

R denotes the recommendation or preference list for vertex

t_j denotes the member of set T which is a list of available items.

N(V) is a neighborhood of vertex v

For experiments, authors use both synthetic and real world datasets. The datasets of Astro Physics, Condense Matter and the DBLP collaboration network are used for experiments.

In addition to graph theory based approaches, there are other data mining and machine learning approaches for *influence maximization problem.* One such approach is discussed in the paper [14]. The authors in this paper propose a system called *Influence Rank* which is used to measure the influence on Twitter users. The system uses a regression based machine learning technique with *Influence Rank* being used as a predictor variable. A set of direct and derived features are used to build the regression model. The system uses *nusupport vector machine (nu-SVM)* for support vector regression. *nu-SVM* allows us to specify an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors that the model can learn, through the parameter nu. The model takes into consideration the direct features like the count of followers, re-tweets for a user and the count of public lists of which a user is a part of. For the derived features, a ratio of count of tweets to the total tweets, a ratio of count of re-tweets to the total re-tweets and a a ratio of count of followers to the total followers. After the feature extraction, label generation is performed on the model. In this process, a system carries out a survey giving preferences to the influencers. After this step, *InfluenceRank* algorithm is then run on the output obtained from the label generation phase, to rank the influencers. This paper uses Twitter dataset for the experiments.

The importance of a user in a network is an important parameter, to determine its potential to influence others. The paper [17] describes a methodology to determine the importance and influence of a Twitter account in a social network. The method also computes the maximization of diffusion of information in the network. In order to detect the influencers in a network, a parameter called *Influence Metric* is proposed in the paper. It is dependent on the followers to followee ratio as well as the number of tweets generated by user accounting for user passivity. It is given by below formula .

$$InfluenceMetric = \frac{tweets_k}{Days_{sincek_{th}tweet}} * OOM(Followers) * log_{10}(\frac{Followers}{Following} + 1) \quad (8)$$

For information diffusion we calculate the Tweet Transmission Ratio which depends on the probability of followers generating tweets which includes both new tweets as well as re-tweets. The Tweet Transmission measurement is given by.

$$TweetTransmission = \frac{TCR_{n+1}}{TCR_n} * RT_{n+1} \quad (9)$$

where,

TCR = Tweet Creation ratio

RT = Retweet

Overall Framework:

• The process starts with selecting the Twitter user for the experiments.

• In the second phase, Twitter characteristics are retrieved.

• In phase 3, the followers are classified in two categories based on *Influence Metric* and the absolute amount of followers each user has. This is followed by sorting categories in descending order to select the top k users from each category.

• For the selected users, steps 2 and 3 are repeated. This process is continued until a specified distance, threshold (layers), between Twitter users is reached. A layer graph is constructed at each step.

• During the 5th phase each of the two generated networks is terminated by an ending vertex (sink). This vertex is connected with all the users-followers of the last layer.

• In the last phase, we discover all the paths, starting from the initial identified user and ending to the sink and consists of exactly 4 steps. *Tweet Transmission Ratio* is calculated across these paths which helps to determine the influence spread.

The data mining based methods can also be used to model the solution for *influence maximization problem.* One of the methods is described in the paper [1]. The authors in this paper, study the software social networks (SSN) Github, to detect the influencers in the network. To measure the influence, three parameters are considered namely number of followers, number of forked projects and number of watched projects. The paper also estimates the spread of influence of the *seed set* of influencers across multiple programming languages. Below are the steps of the framework discussed in the paper.

The first step is the parameter analysis phase. Using *Spearman correlation*, we de-

termine if the three parameters imply same notion of influence.
In the second step, we observe the behavior of top 30 Git Hub users. This behavior is analyzed in terms of the above mentioned three parameters.
In the last phase, we study top 10 programming languages used by Git Hub users. Each language is associated with two metrics, *u.fork* and *u.watch*. We compute these metrics per user for top 10 programming languages. Users are sorted based on their language specific metrics. From this investigation we study if the influential Git Hub users extend their influence over multiple languages.

In a social network, a user may consume the information from others in the network. However, a user may choose to not propagate this information across the network. This user is termed as a passive user. Some of the research works, consider *passivity* of a user while building a solution for *influence maximization*. The paper [18] proposes the *Influence Passivity (IP)* algorithm which measures the passivity of user to determine the influencers in social networks. IP algorithm assigns a relative influence score and a passivity score to every user. The algorithm uses below assumptions:
1. A user's influence score depends on the number of people influenced and their passivity.
2. A user's passivity score depends on the influence of the exposed users but is not influenced by it. It also depends on how much the user rejects other user's influence scores simultaneously. The authors use Twitter dataset with tweets consisting of urls for experiments.

Thus, we summarized some of the existing work for the *influence maximization problem*. We compare our proposed algorithms for *influence maximization* with the greedy approximation solution. In chapter 4, we describe the proposed methodology for *influence maximization* problem.

CHAPTER 4

Methodology for Influence Maximization

As a part of this project, I studied the greedy approximation algorithm. However, the greedy algorithm has a large computation time and is not suitable for large graphs. As an improvisation of the greedy algorithm, CELF algorithm was proposed in paper [13] which runs 700 times faster and achieves results comparable to that of greedy algorithm. However, the initialization step of CELF algorithm involves computing marginal gain of all the vertices in a graph. This is expensive for large graphs. In this project, we modify the CELF algorithm and compare its results with the existing models. Also, for undirected graphs we propose an acyclic spanning graph based algorithm for *seed set detection.* Once the *seed set* is detected, we compute the number of vertices influenced by it using a Linear Threshold model.

4.1 Greedy Approach

The *Influence Maximization* problem is a NP-hard problem. Kempe et al. in their work [11] propose an approximation greedy approach to solve the *influence maximization* problem. The greedy solution works by selecting one vertex at a time and estimating the number of other vertices in the network which it influences. This process is termed as spread estimation process. We use Linear Threshold model to compute the spread estimation of a vertex. In this model each vertex is assigned a threshold value. If the sum of the edge weights of incoming active vertices is greater than the current vertex threshold, then it becomes active. We assign the edge weights as (1/in-degree) for a non-zero value of in-degree. If the in-degree is zero then, the edge weight is zero. The greedy algorithm is an approximation solution with an approximation of (1 - $\frac{1}{e}$ - ϵ). It gives effective results, however, it is computationally slow as it computes the spread vertex by vertex. Hence, it is not suitable for large graphs. The pseudo code for Greedy algorithm is given in Algorithm 2.

Algorithm 2 Greedy Solution

```
Input:  K size of seed set, f monotone submodular function
Output:  seed set of influencers
1: procedure GREEDY SOLUTION($g, K, f$)
2: Initialization: $S$ <- $\phi$
3:     for  $i$ = 1 to $k$ do
4:         u ← $argmax_{w \in \frac{V}{S}}((f(u \cup S) - f(S))$
5:         S ← $S \cup u$
6:     return S
```

For the spread estimation process Linear Threshold model is used. In this model, each vertex is associated with a threshold value. A vertex is influenced, if the sum of its incoming active neighbors is greater than the threshold. Once influenced, a vertex is said to be in active state. Also, once a vertex is activated it remains in the active state. The model stops when no more vertices can be added to the influencer set. The pseudo code for Linear Threshold Model is given in Algorithm 3.

Algorithm 3 Linear Threshold

```
Input:  K size of seed set, seed set, g, vertex-threshold
Output:  Set of influenced vertices
 1: procedure COMPUTE-K-DIFFUSION($g, K, seedset, vertex - threshold$)
 2: Initialization: total-k-influenced <- $\phi$ , curr-active-nodes <- seed-set, total-influenced <- seed-set
 3:     while True do
 4:         total-k-influenced $\leftarrow$ compute-curr-diffusion(k, g, curr-active-nodes, vertex-threshold)
 5:         if Length(total-k-influenced) equals Length(total-influenced) then
 6:             break
 7:         total-influenced $\leftarrow$ total-k-influenced
 8:         curr-active-nodes $\leftarrow$ total-k-influenced

 9: procedure COMPUTE-CURR-DIFFUSION($g, K, active - set, vertex - threshold$)
10: Initialize curr-active-list <- active-set
11:     for i = 0 to Length( active-set ) do
12:         for n in neighbors of current vertex do
13:             Compute sum of edge weights of all active incoming neighbors
14:             if sum greater than threshold of vertex then
15:                 Add vertex to active vertex list
16:     return curr-active-list
```

Example: If we execute the greedy Algorithm 2 on the graph in Figure 1 with the *seed set* size (k = 2), we get the influencer vertices as 0 and 8. The vertices influenced by this seed set are 0, 8, 9, 10 and 11.

The graph in Figure 3 is the output graph of the Algorithm 2. In the output graph the vertices in red color are the influencer vertices whereas, the vertices green in color represent the influenced vertices.

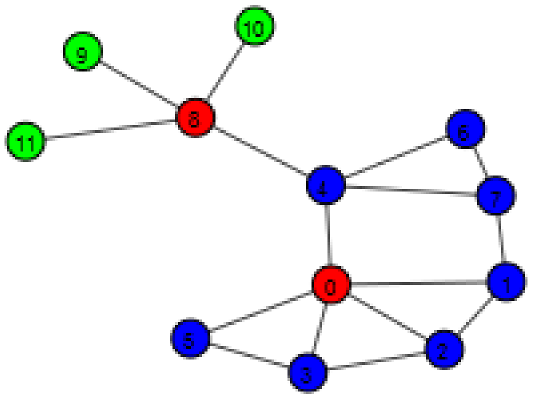

Figure 3: Example: Greedy algorithm output graph

4.2 Influencers Seed Set Detection Using Acyclic Spanning Graph

The centrality of a graph describes the importance of a vertex. Some of the types of centralities are degree centrality, closeness centrality, betweeness centrality etc. PageRank centrality defines the importance of a vertex based on the importance of other vertices it is connected to and the vertices that are connected to it. As PageRank centrality considers importance of a vertex in computation, in this approach, we use the PageRank centrality to compute the *seed set* of influencers in an undirected graph. One of the steps in this approach is the construction of an acyclic spanning graph from the given graph. The given undirected graph, consists of self-loops and cycles. These are known as the feedback edges. The construction of acyclic spanning graph removes these feedback edges thereby increasing the performance of seed set detection. The paper [6] uses a similar approach where, the authors use Closeness centrality measure in their works. We use PageRank centrality as a measure to find the most central vertices because PageRank centrality is computed considering the importance of a vertex.

The steps followed in this approach to find the *k-influencer seed set* are as below:

1. Compute the most central vertex using PageRank centrality in the original graph.

2. Construct an acyclic spanning graph from the original graph by using the most central vertex as the starting vertex of the graph.

3. Compute the k most central vertices using the PageRank centrality from the spanning graph. These form the *k- influencer seed set.*

The pseudo code for acyclic spanning graph based algorithm for the influencers seed set detection is given in Algorithm 4.

Algorithm 4 Influencers Seed Set Detection using Spanning Graph

```
Input:  K size of seed set, graph G
Output:K Seed Set of influencer vertices
 1: procedure SEED SET DETECTION($G, K$)
 2:     Compute PageRank centrality of all vertices in G
 3:     $SpanningGraph \leftarrow$ Spanning Graph with most central vertex as starting vertex
 4:     Compute PageRank centrality of all vertices in SpanningGraph
 5:     $seed\ set \leftarrow$ K-most central vertices in SpanningGraph
 6:     return seed set

 7: procedure CONSTRUCT ACYCLIC SPANNING GRAPH($G$)
 8:     Initialization: $V_{SG} \leftarrow \phi$, $queue \leftarrow \phi$, $visited \leftarrow \phi$, $E_{SG} \leftarrow \phi$
 9:     for all vertices in graph do
10:         Compute Centrality of vertex C(n)
11:     Begin Vertex $\leftarrow$ Vertex with Maximum centrality
12:     queue $\leftarrow$ Insert Begin vertex
13:     while queue not empty do
14:         $v \leftarrow$ Get vertex from queue
15:         if v not in visited then
16:             visited $\leftarrow$ Insert v in visited
17:         $N \leftarrow$ Neighbors of vertex v sorted in descending order on centrality
18:         for all vertices z $\in$ N(v) and z $\notin$ visited do
19:             $E_{SG} \leftarrow E_{SG} \cup \{v, z\}$
20:             $V_{SG} \leftarrow V_{SG} \cup \{z\}$
21:             queue $\leftarrow$ Insert z in queue
22:     return $SG$
```

Once the *seed set* of influencers is detected, we use the Algorithm 3 to compute the spread of influencers *seed set* i.e the number of vertices influenced by the *seed set* in the social network model.

Consider the input graph in Figure 1. If we compute the influencers *seed set* of size (k = 2) using Algorithm 4 we get the influencer vertices as 0 and 8. The vertices influenced by this seed set are 0, 8, 9, 10 and 11.

Figure 4 shows the spanning graph constructed from the original graph.

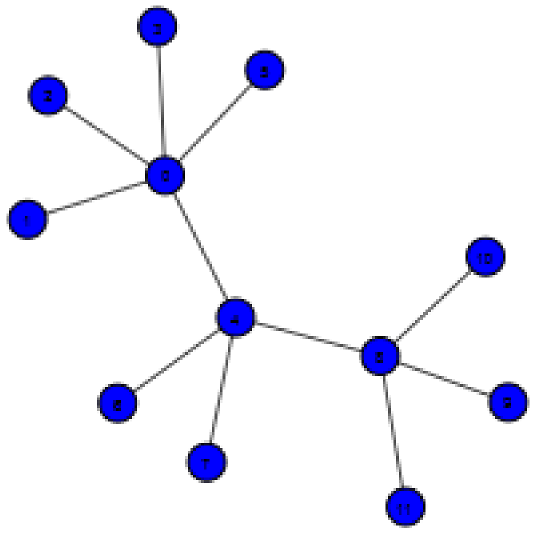

Figure 4: Example: Output Spanning graph

The graph in Figure 5 is the output graph of the Algorithm 4 showing the influencers in the red color and the green vertices represent the vertices influenced by the *seed set*.

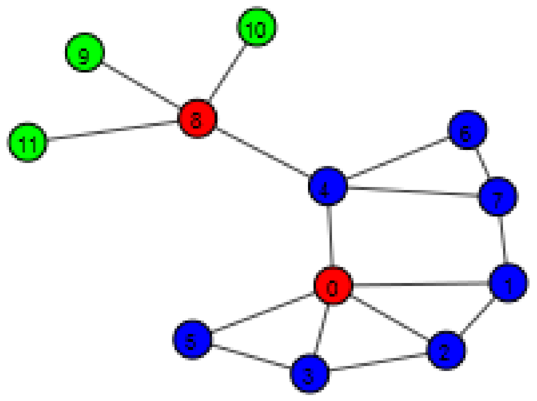

Figure 5: Example: Spanning Graph algorithm output graph with influencers

4.3 Cost Effective Forward Selection Greedy Algorithm (CELF)

This algorithm is an improvisation of the greedy approximation algorithm [13]. It uses the properties of submodularity and Lazy forward selection to reduce the number of spread estimation calls for the vertices of a graph.

A function f is said to be a submodular function if it satisfies the property that the marginal gain obtained by adding a single element to a set S will decrease for a larger value of set size. Thus, submodular functions satisfy the diminishing returns property [22]. It can be stated as,

$$f(S \cup \{u\}) - f(S) >= f(T \cup \{u\}) - f(T) \tag{10}$$

where, S $\subset$ T

In the greedy Algorithm 2, consider that in the current iteration the *seed set* detected is S and for vertex $v \in$ V\ S the computed spread is f (v | S). Let T be the *seed set* detected in the previous iteration of the algorithm such that T $\subset$ S. For vertex u, in the previous iteration $u \in$ V\ T the computed spread is f (u | T). Also,

f (v | S) >= f (u | T). Hence, by the property of submodularity ,

$$f(u|S) <= f(u|T) <= f(v|S) \tag{11}$$

Lazy forward selection is based on the submodularity property. We implement it by maintaining two properties per vertex namely, the marginal gain and the iteration during which the marginal gain was calculated. A priority queue is used to store the vertices such that the vertex with the highest marginal gain will be the first element of the queue. Initially, we compute the marginal gain of all vertices and insert them in the queue. In the subsequent iterations, the top element of the queue is extracted. If the iteration of the extracted vertex is the same as the current iteration then it implies that it is the vertex with the maximum marginal gain for the current *seed set* and hence is added to the *seed set.* If the iteration is lower than the current iteration, then the marginal gain of the vertex is computed for that vertex considering the current *seed set* S and is inserted in the priority queue. Hence, we reduce the spread estimation calls with forward selection. The pseudo code for CELF algorithm is given in Algorithm 5.

Algorithm 5 CELF

```
Input:  K size of seed set, f submodular function, graph G
Output:  Seed set of influencers
 1: procedure CELF Seed Set Detection(G)
 2:     Initialization: $S \leftarrow \phi$, Priority Queue $Q \leftarrow \phi$, iteration $\leftarrow$ 1
 3:     for vertex u in vertices do
 4:         u.mg $\leftarrow$ f(u|$\phi$)
 5:         u.iteration $\leftarrow$ 1
 6:         Insert vertex u in Q
 7:     while iteration <= K do
 8:         Delete top element v from Q
 9:         if v.iteration equals iteration then
10:             S <- S $\cup$ {v}
11:             iteration <- iteration + 1
12:         else
13:             v.mg $\leftarrow$ f(v|S)
14:             v.iteration $\leftarrow$ iteration
15:             Insert v in Q
16:     return S
```

Once the *seed set* of influencers is detected, we use the Algorithm 3 to compute the spread of influence i.e the number of vertices influenced in a network. If we run the Algorithm 5 on the input graph in the Figure 1 with *seed set* size (k = 2) we get the influencer vertices as 8 and 2. The vertices influenced by this *seed set* are 8, 2, 9, 10 and 11.

Figure 6 is the output graph of the *seed set* detection Algorithm 5 and spread estimation Algorithm 3. The vertices in the red color are the influencer vertices. The vertices green in color represent the vertices influenced by the *seed set.*

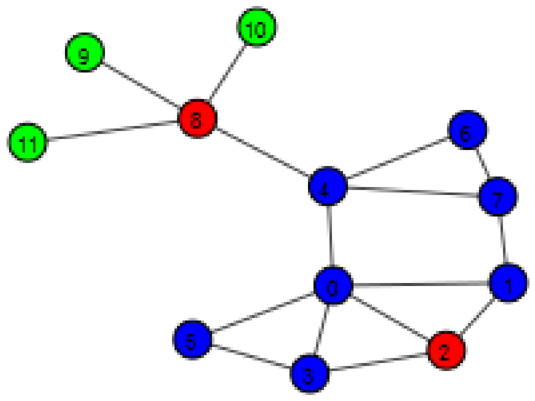

Figure 6: Example: CELF Output graph with influencers

4.4 Modified CELF algorithm

In Algorithm 5, in the first step, the marginal gain for all the vertices is computed. In the case of large graphs, computing marginal gain for all the vertices will deteriorate the performance of the algorithm.
In order to reduce the computation time at the preliminary step, we compute the vertex cover of the given graph. In the first step of the Algorithm 6, the marginal gain of only the vertices in vertex cover is computed. Thus, we reduce the marginal gain computations in the preliminary step of algorithm. To compute the vertex cover of the input graph, we use a greedy approximation solution. The pseudo code for the modified CELF algorithm is given in Algorithm 5.

Algorithm 6 CELF With Vertex Cover

```
Input:  K size of seed set, f submodular function, graph G
Output:  Seed set of influencers
 1: procedure CELF Seed Set Detection($G$)
 2:     Initialization: $S \leftarrow \phi$, Priority Queue $Q \leftarrow \phi$, iteration $\leftarrow 1$, vertex cover $\leftarrow \phi$
 3:     vertex cover $\leftarrow GetVertexCover(G)$
 4:     for vertex u in vertex cover do
 5:         u.mg $\leftarrow$ f(u|$\phi$)
 6:         u.iteration $\leftarrow$ 1
 7:         Insert vertex u in Q
 8:     while iteration <= K do
 9:         Delete top element v from Q
10:         if v.iteration equals iteration then
11:             S <- S $\cup$ {v}
12:             iteration <- iteration + 1
13:         else
14:             v.mg $\leftarrow$ f(v|S)
15:             v.iteration $\leftarrow$ iteration
16:             Insert v in Q
17:     return $S$

18: procedure GetVertexCover($G$)
19:     G-undirected $\leftarrow$ Get Undirected Graph from G
20:     degree-vertex $\leftarrow$ Map of vertex, degree sorted in the descending order by degree
21:     for vertex u in degree-vertex do
22:         neighbors $\leftarrow$ Neighbors of vertex u
23:         if $u$ not in vertex cover then
24:             for vertex n in neighbors do
25:                 if $n$ not in vertex cover then
26:                     vertex cover $\leftarrow$ vertex cover $\cup$ u
27:                     break
```

If we execute Algorithm 6 on the graph in Figure 1, we get the results same as that of Algorithm 5. The vertex cover includes the vertices 0, 1, 2, 3, 4, 7, 8.

The chapter 5 summarizes the experiments and results obtained by execution of different algorithms on synthetic and real world datasets.

CHAPTER 5

Experiments

Synthetic as well as real world datasets are used for performing the experiments. For the preliminary testing of our models, random graphs were constructed using the Barabasi random graphs model [2]. We use the Python igraph library for various social network graph operations [16]. The section 5.1 describes the experiments performed with synthetic random graphs. The experiments performed with various real world social network datasets are described in the sections 5.2, 5.3, 5.4, 5.5 and 5.6 .

5.1 Random Graphs

For experiments, random graphs are constructed using the Barabasi random graphs model. The random graph constructed is an undirected graph, consisting of 78 vertices and 153 edges. We use the Algorithm 4 to detect the *seed set* of size 10 *(K = 10)*. We get the influencer *seed set* of vertices,
1, 3, 0, 4, 9, 5, 18, 24, 27, 26
On computing the number of vertices influenced by the *seed set* using Algorithm 3, we see that 62 vertices in the graph are influenced by the *seed set.* Figure 7 represents the output random graph. The vertices in red are the *seed set* vertices whereas the vertices green in color are the vertices influenced by the *seed set.*

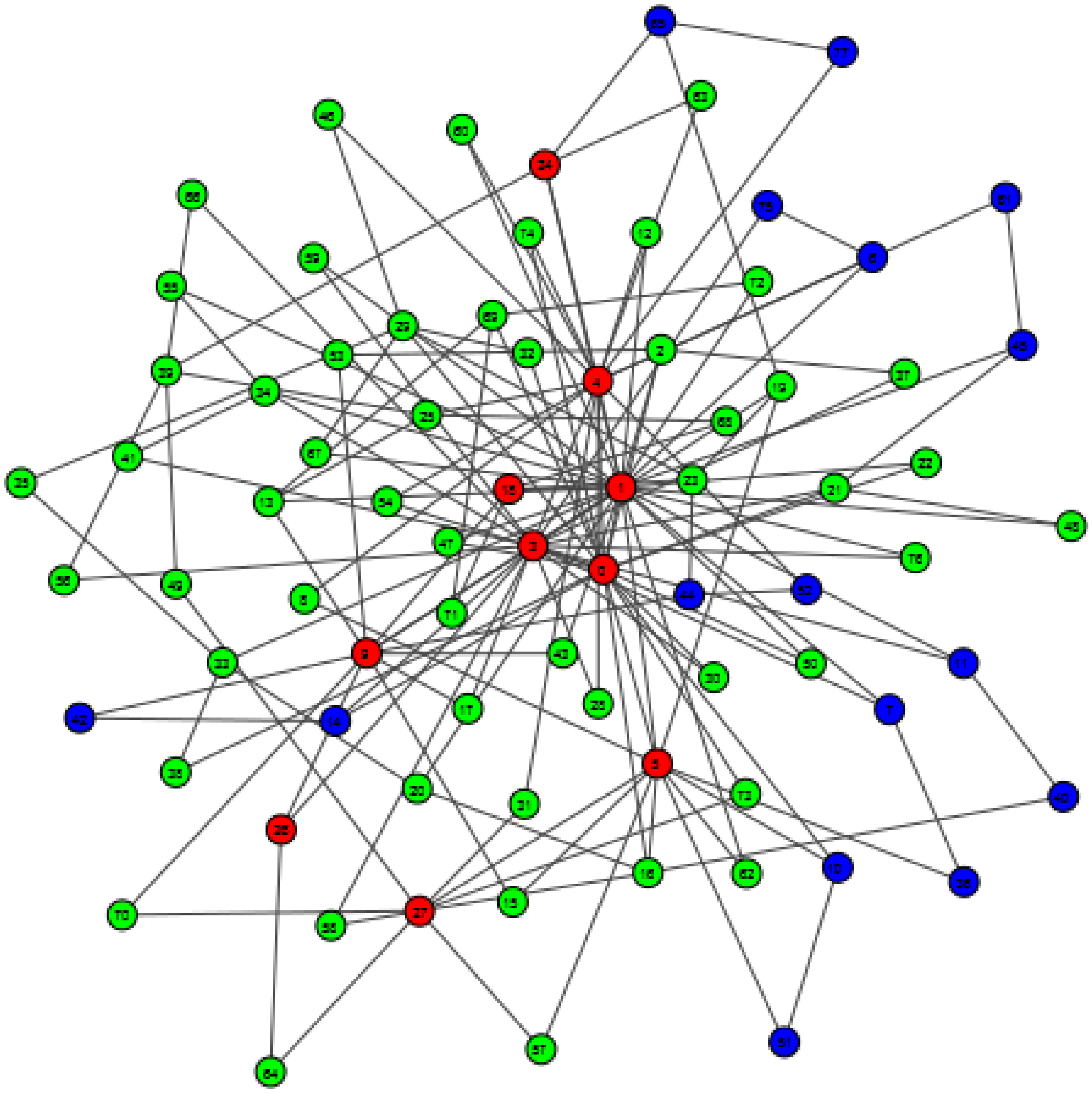

Figure 7: Barabasi Random Graph: Output Graph

In addition, the results are also compared against the greedy *seed set* detection Algorithm 2. The *seed set* consists of vertices

0, 1, 3, 4, 6, 7, 10, 11, 26, 27

On computing the number of vertices influenced by the *seed set* using Algorithm 3 we see that 70 vertices in the graph are influenced by the *seed set.* Figure 8 represents the output of the greedy Algorithm 2.

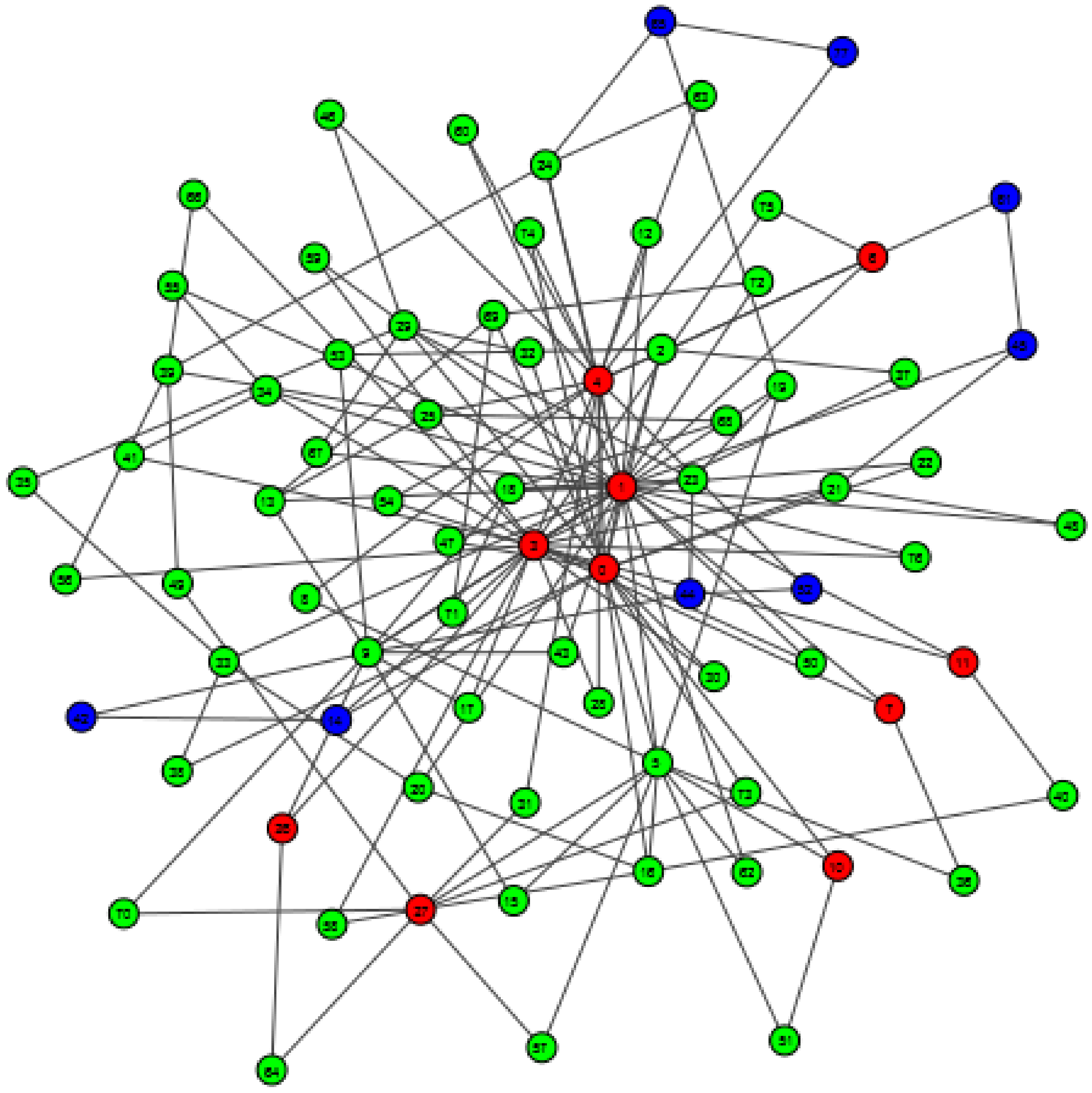

Figure 8: Barabasi Random Graph: Greedy Output Graph

5.2 Zachary's Karate Club Dataset

The social network graph for Zachary's Karate Club, is undirected in nature and contains 34 vertices and 78 edges [21]. This dataset was compiled by observation of friendships among members of club for over two years [7]. During the period of observation, there was a conflict between the administrator and instructor of the club which lead to splitting of the club in two halves. We use the Algorithm 4 to detect *seed set* of size 2 *(K = 2)*. With this we get, the influencers as vertices 0 and 33 which is in correspondence with the actual known values.
On computing the number of vertices influenced by the *seed set* using Algorithm 3 we see that all the vertices in the graph are influenced by the *seed set*. Figure 9 represents the social graph of Karate club dataset. The Karate social network graph is undirected and unweighted in nature.

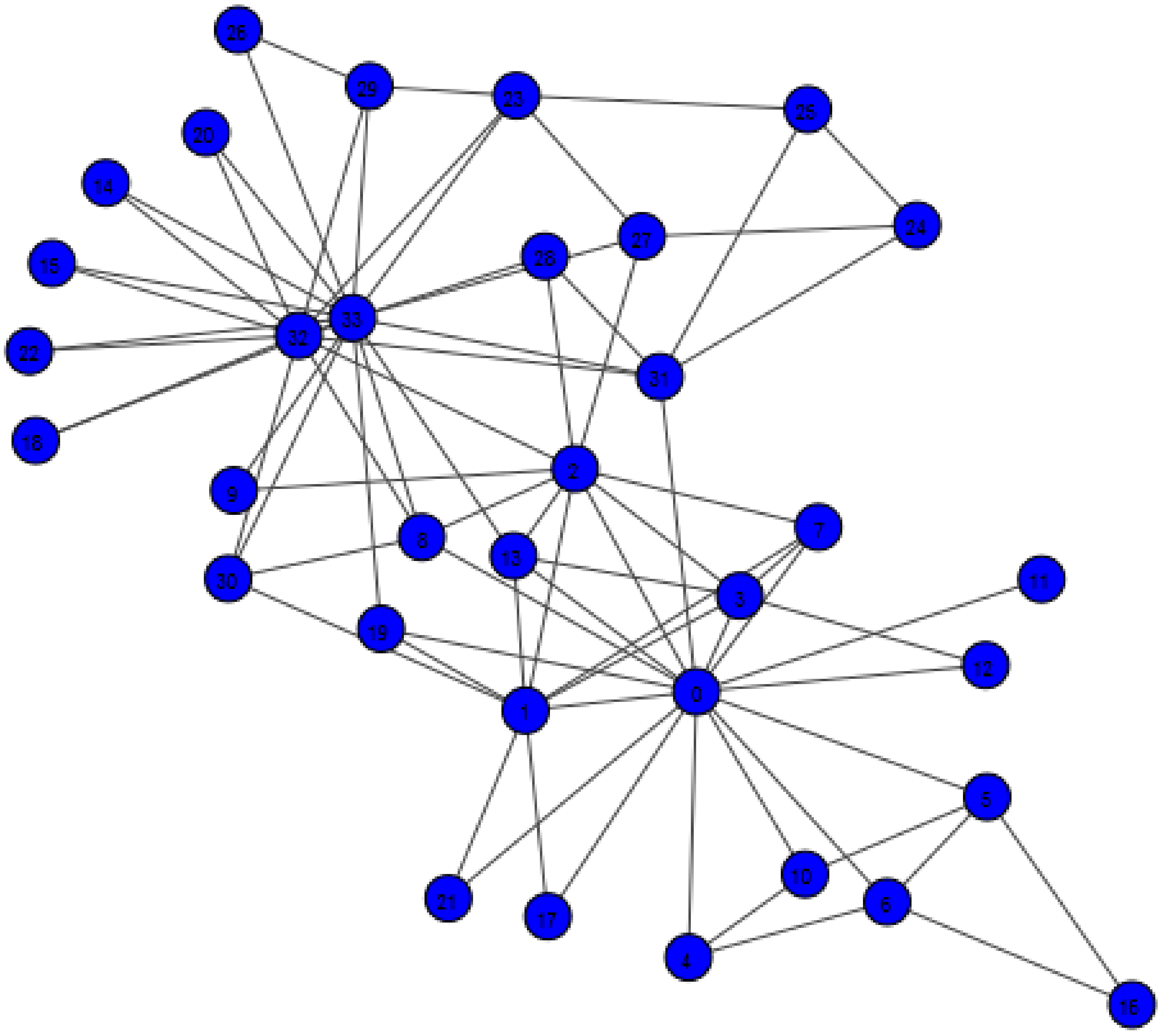

Figure 9: Zachary's Karate Club Social Network Graph

While computing influencers *seed set* using Algorithm 4, a spanning graph from the original graph is constructed. Figure 10 shows the acyclic spanning graph for Zachary's Karate club dataset. This spanning graph is constructed using 33 as the starting vertex.

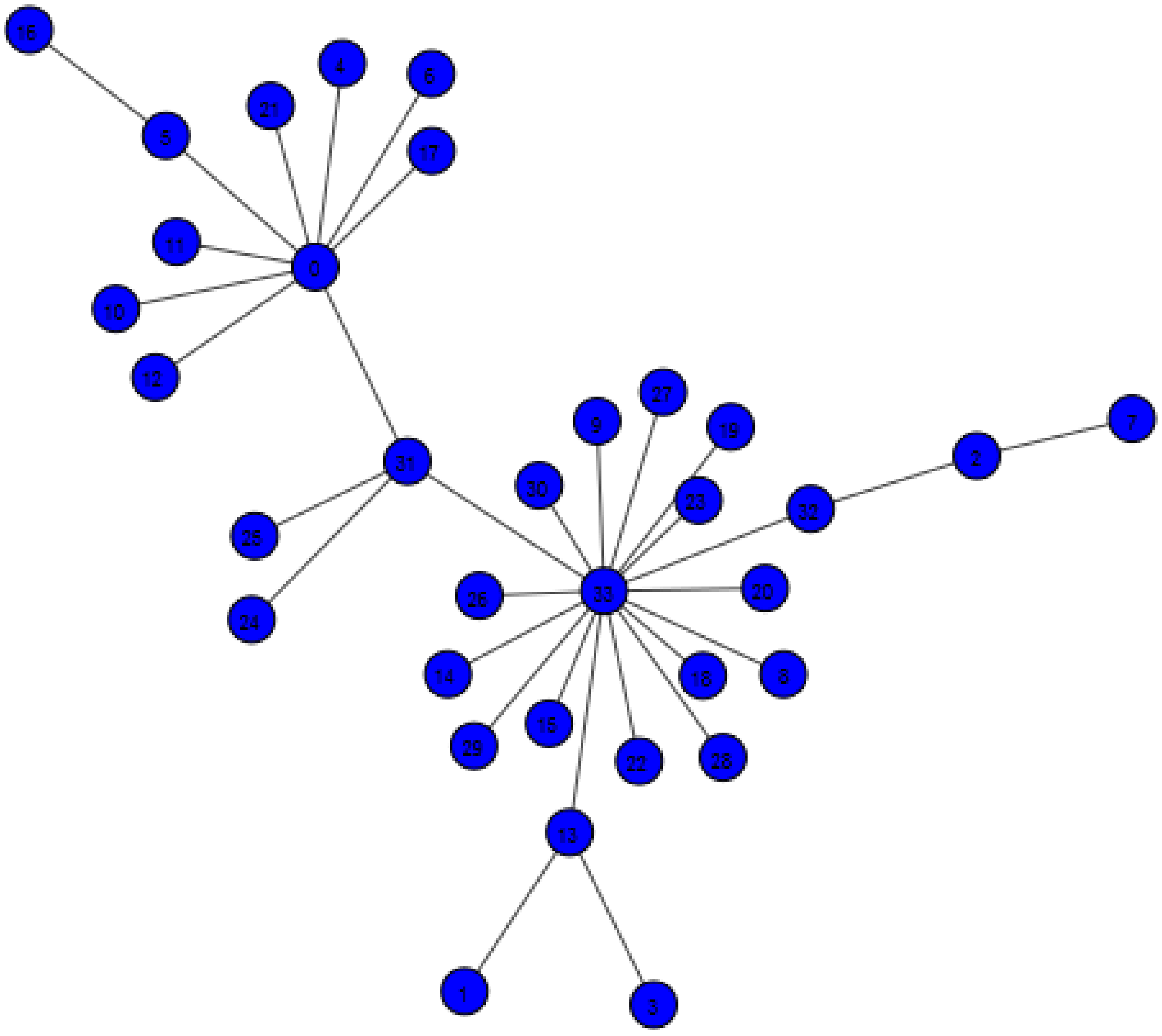

Figure 10: Zachary's Karate Club Social Network: Acyclic Spanning graph

Figure 11 shows the output graph of *seed set* detection Algorithm 4 and spread estimation Algorithm 3, where the vertices in red are *seed set* of influencers and the vertices green in color represent the number of vertices influenced by the *seed set* of influencers.

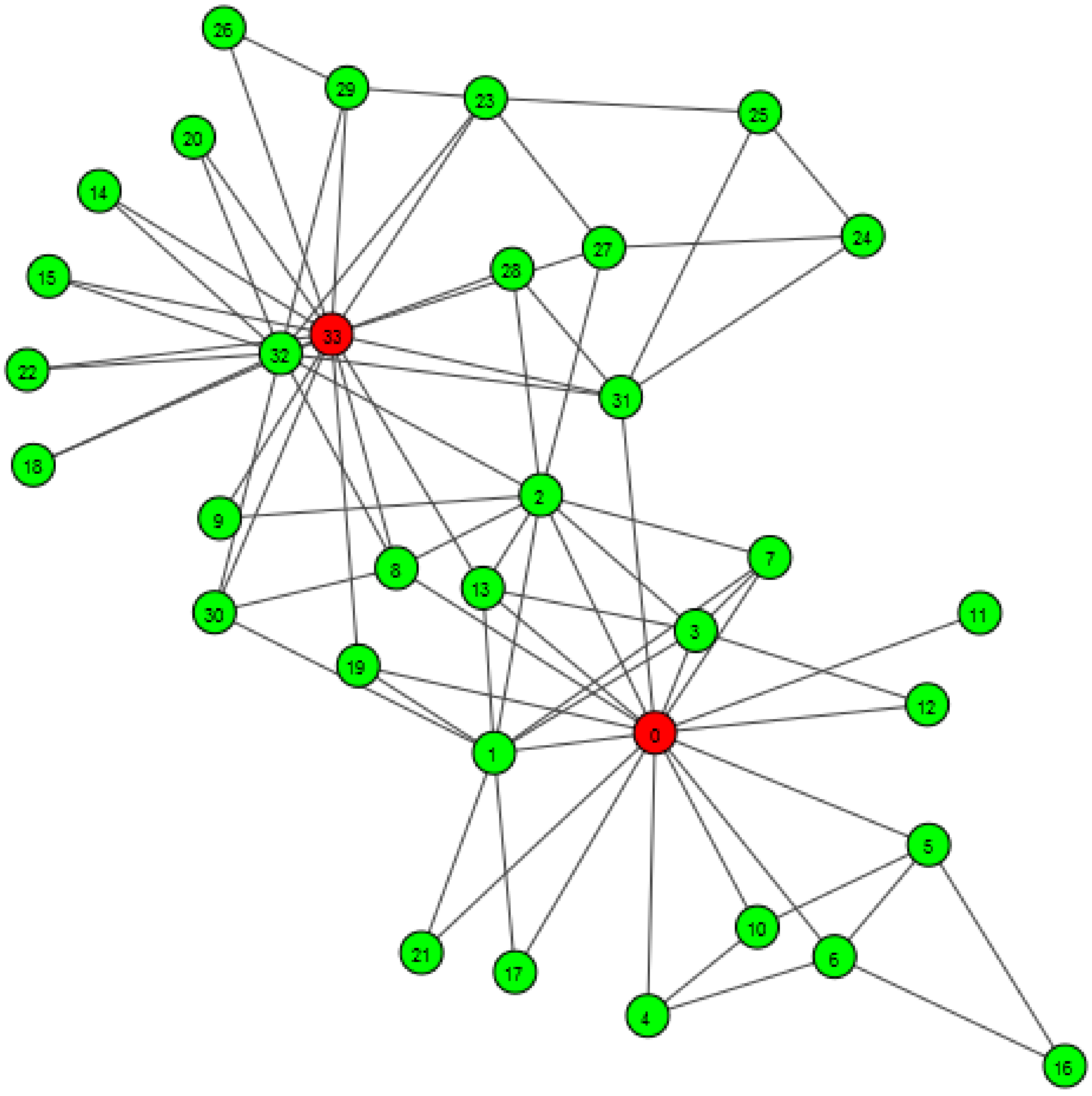

Figure 11: Zachary's Karate Club Social Network: Output graph

In addition, the results are also compared against the greedy *seed set* detection Algorithm 2. The Algorithms 4 and 2 give the same results with vertices 0 and 33 as*seed set* of influencers. This *seed set*, influences all the other vertices in the graph.

5.3 Novel co-appearance network

This is a co-appearance network of characters in Victor Hugo's novel "Les Miserables". The characters in the novel form the vertices of the graph. Two characters who co-appeared in the same chapter of novel are connected with the edges. This data set is obtained from [12]. The graph is an undirected network with 77 vertices and 254 edges.
We use Algorithm 4 to detect the *seed set* of size 7 *($K = 7$).* The output of Algorithm 4 is the influencer *seed set* of characters *Valjean, Gavroche, Myriel, Fantine, Marius, Thenardier* and *MlleGillenormand.*
On computing the number of vertices influenced by the *seed set* using Algorithm 3 we see that 75 vertices in the graph are influenced by the *seed set.* Figure 12 represents the co-appearance social network.

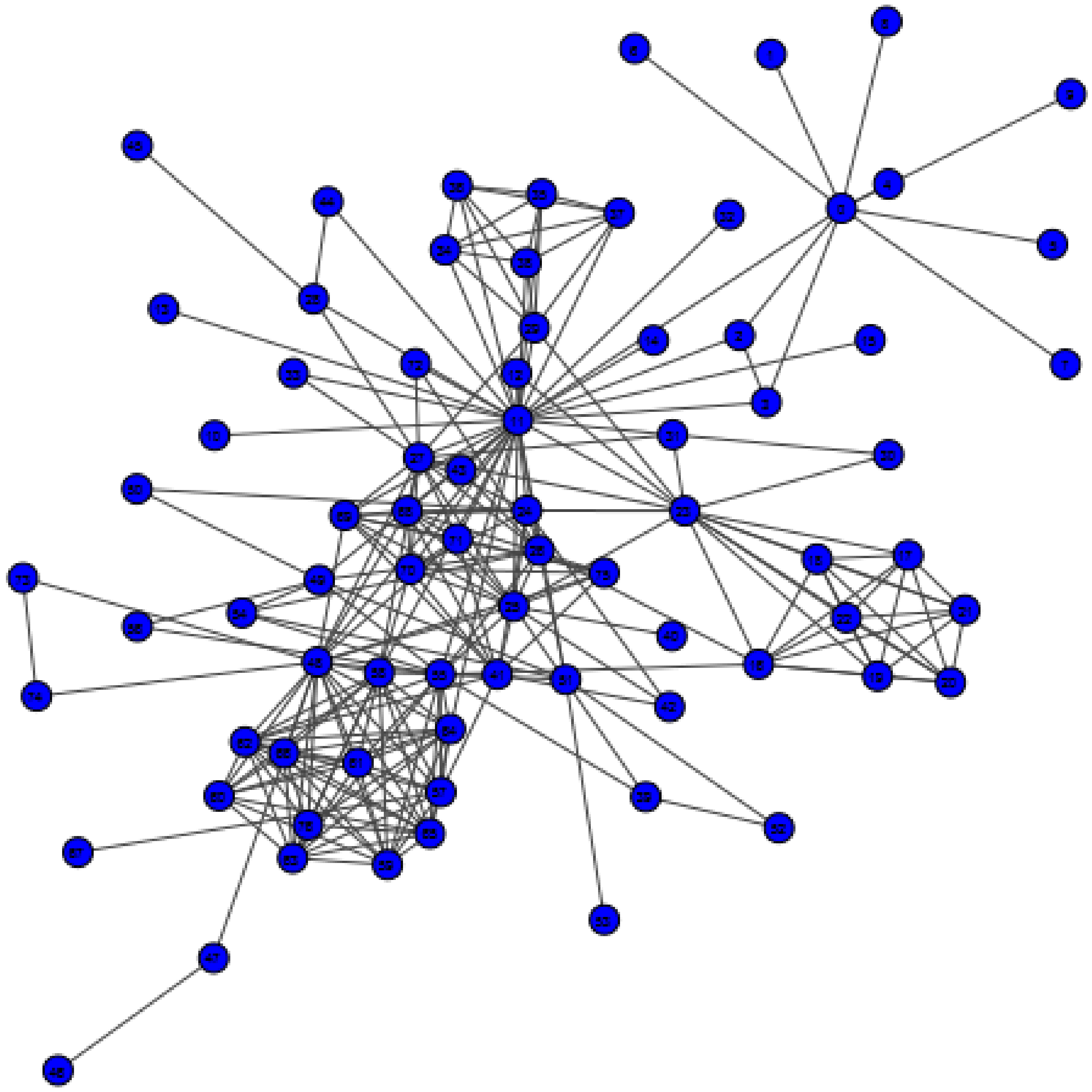

Figure 12: Novel Co-appearance Network

While computing influencers *seed set* using Algorithm 4, a spanning graph from the original graph is constructed. Figure 13 shows the acyclic spanning graph for co-appearance network. This spanning graph is constructed using 11 as the starting vertex.

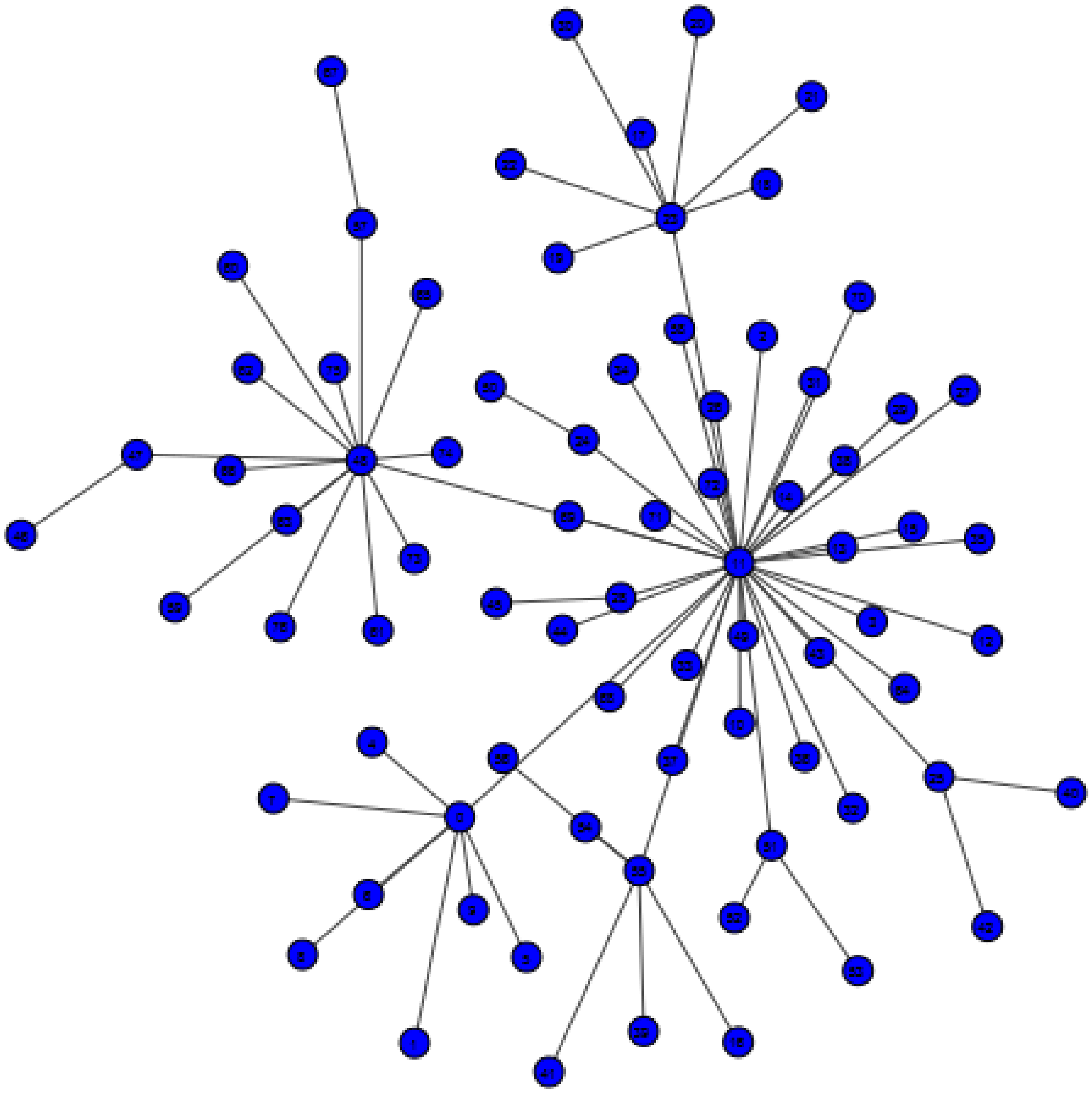

Figure 13: Novel Co-appearance Network: Acyclic Spanning graph

Figure 14 shows the output graph of *seed set* detection Algorithm 4 and spread estimation Algorithm 3, where the vertices in red are *seed set* of influencers and the vertices green in color represent the number of vertices influenced by the *seed set* of influencers.

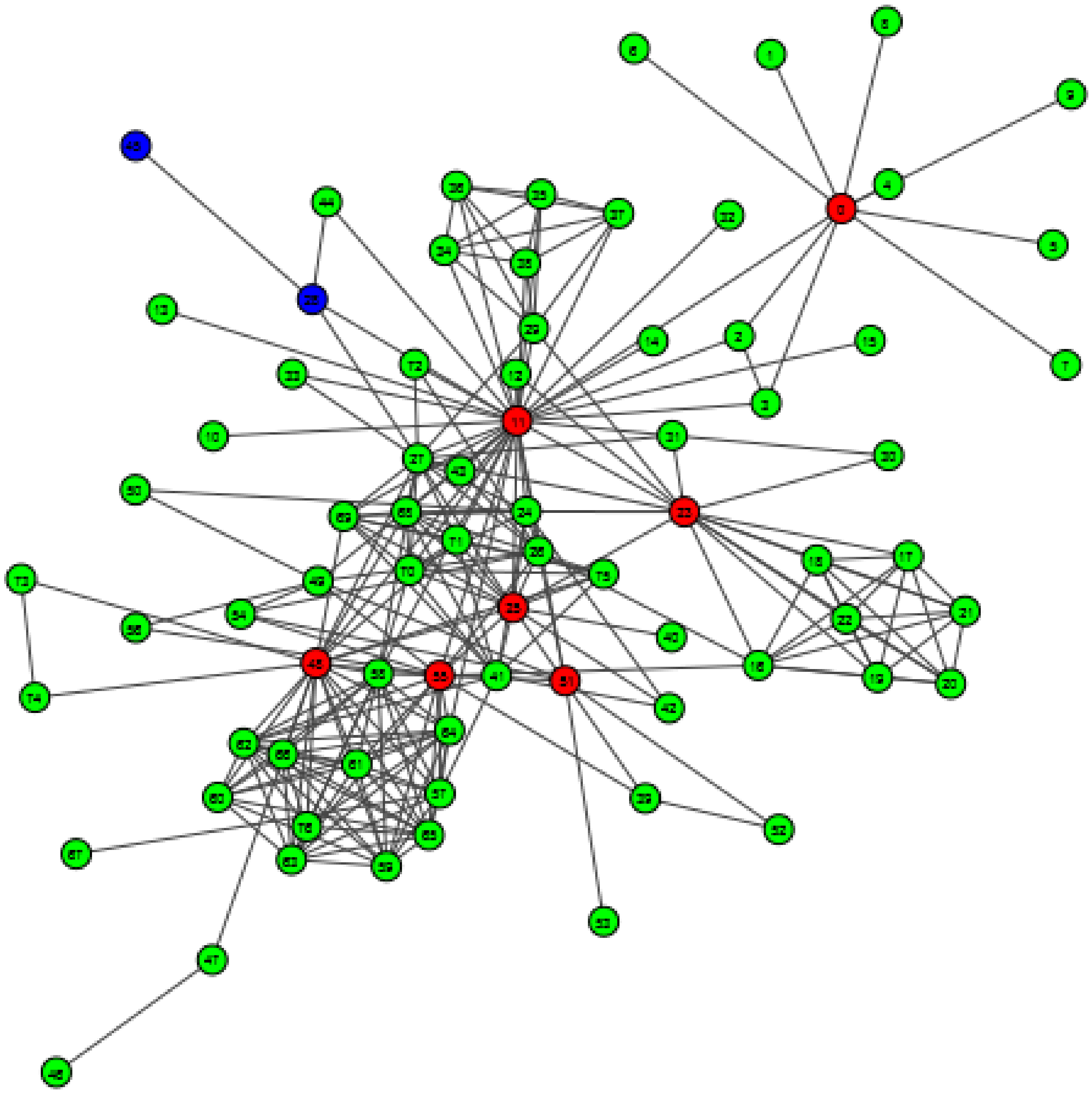

Figure 14: Novel Co-appearance Network: Output Graph

In addition, the results are also compared against the greedy *seed set* detection Algorithm 2. With the greedy Algorithm 2, we get influencer *seed set* of characters *Myriel, Napoleon, MlleBaptistine, MmeMagloire, CountessDeLo, Gavroche and Fauchelevent*. This *seed set*, influences all the other vertices in the graph. Figure 15 represents the output graph of the greedy Algorithm 2.

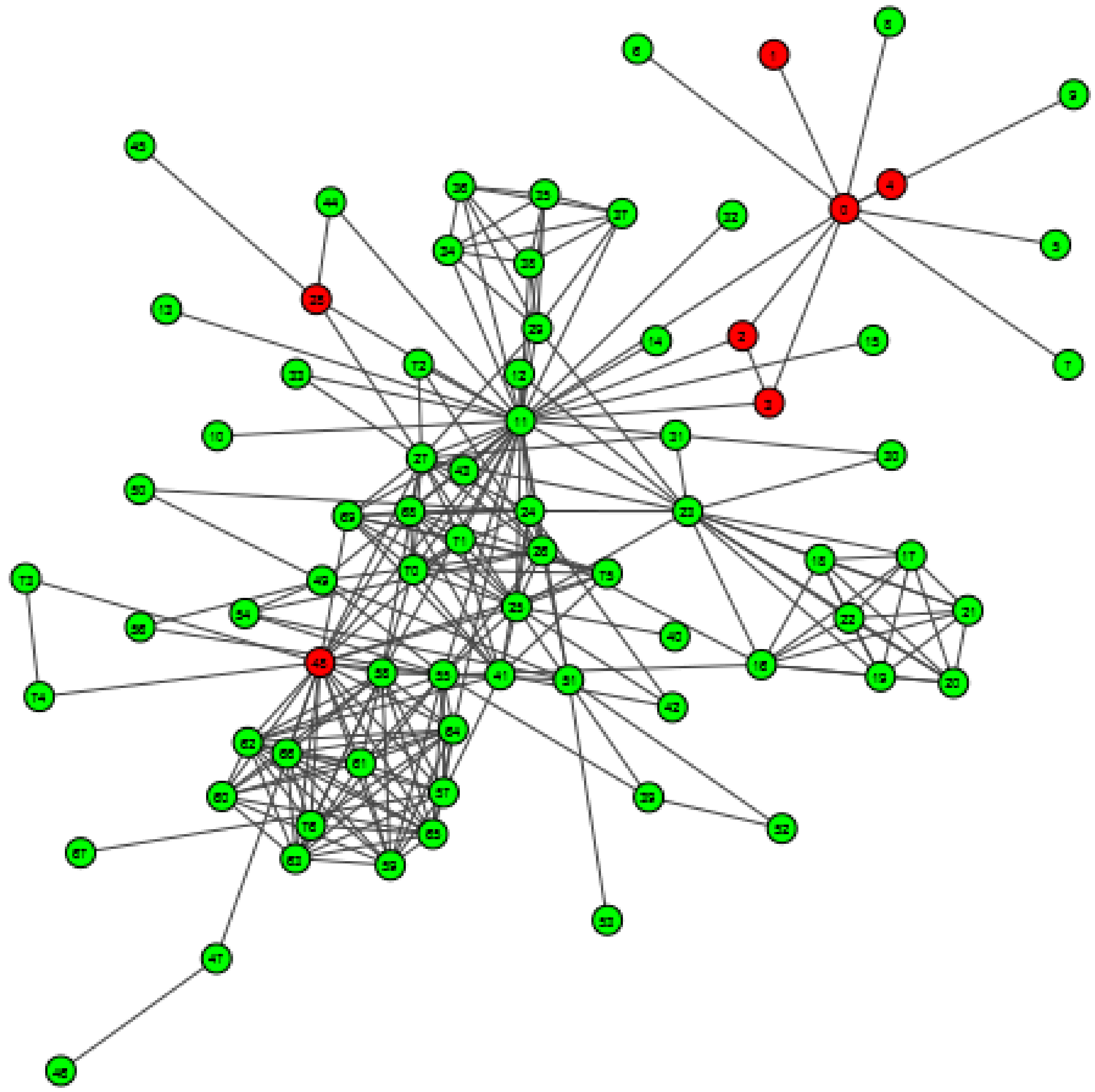

Figure 15: Novel Co-appearance Network: Greedy Output Graph

5.4 Twitter Dataset

We use the Twitter social network dataset from SNAP which is a dataset repository from Stanford University [19]. A subset of the dataset is used to construct a directed social network graph. This graph consists of 236 vertices and 2478 edges. Figure 16 shows the Twitter social network graph.

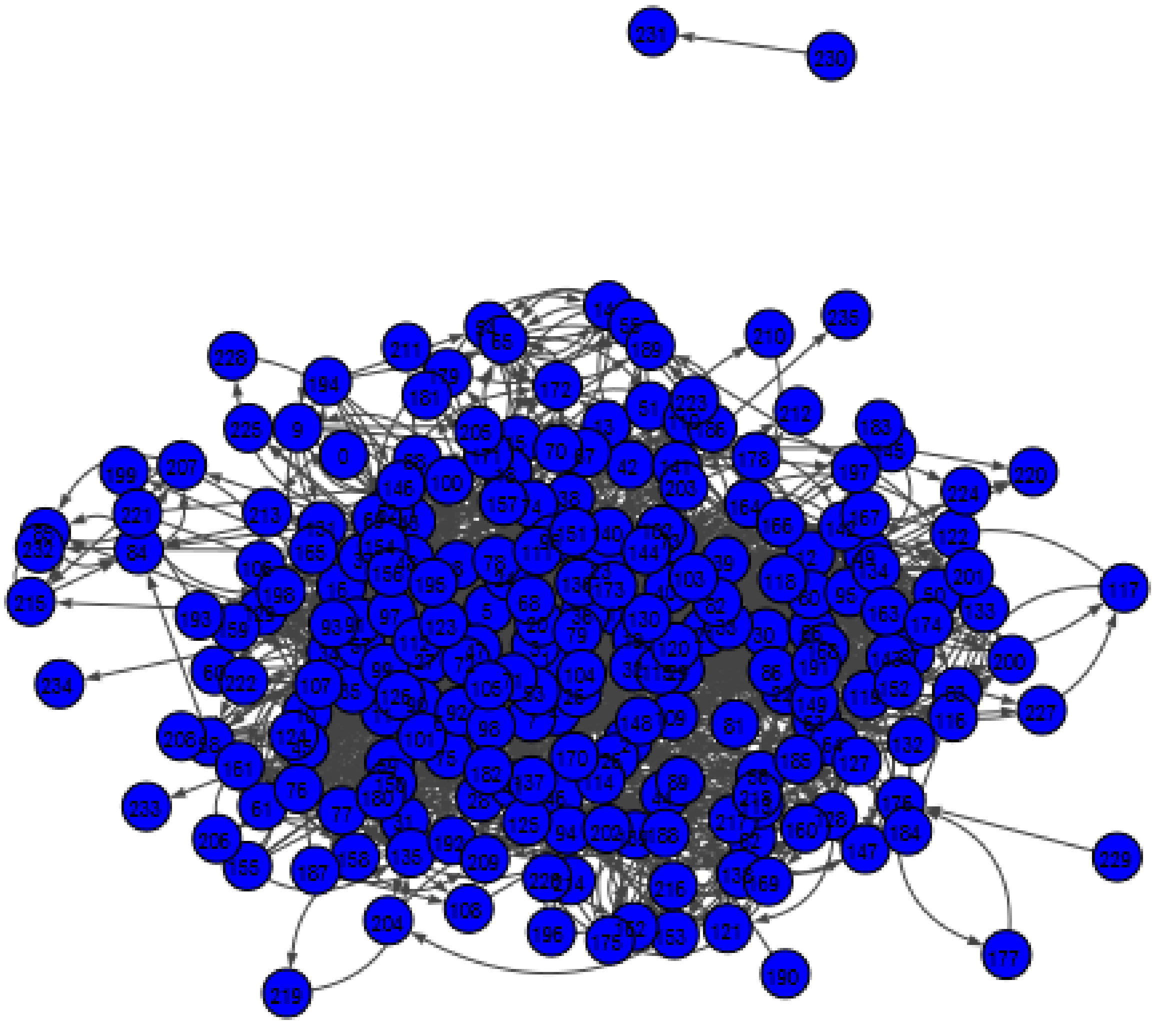

Figure 16: Twitter Social Network Graph

We compute the *seed set* of size 20 using Algorithm 6. In Algorithm 6, we use vertex cover computation as a preliminary step. Also, Algorithm 3 is used to compute the number of vertices influenced by the *seed set.* Figure 17 represents the output Twitter social graph of Algorithm 6. The vertices in red represent the *seed set* of influencers whereas the vertices in green represent the vertices influenced by the *seed set.* In total, 144 vertices are influenced by *seed set* of size 20.

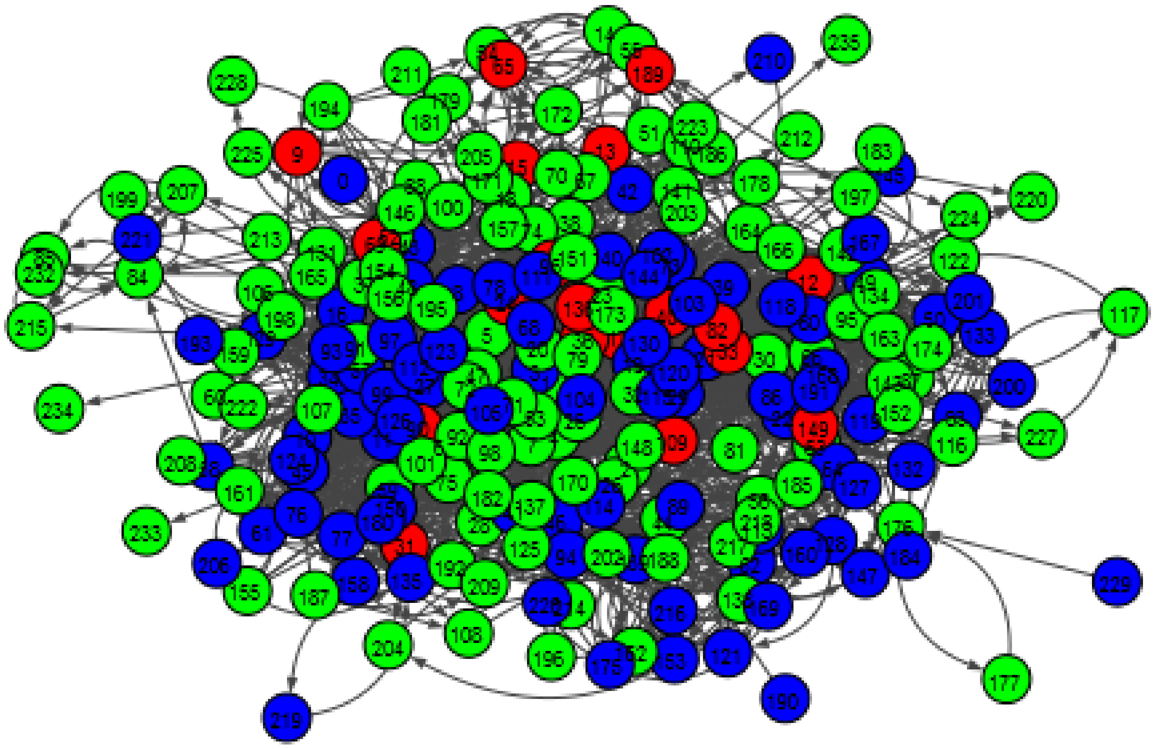

Figure 17: Twitter Social Network: Output CELF with Vertex Cover

Similarly, we compute the *seed set* of size 20 using Algorithm 5, CELF without vertex cover. Also, Algorithm 3 is used to compute the number of vertices influenced by the seed set. Figure 18 represents the output Twitter social network graph of Algorithm

5. The vertices in red represent the influencers whereas the vertices in green represent the vertices influenced by the seed set. In total, 142 vertices are influenced by seed set of size 20.

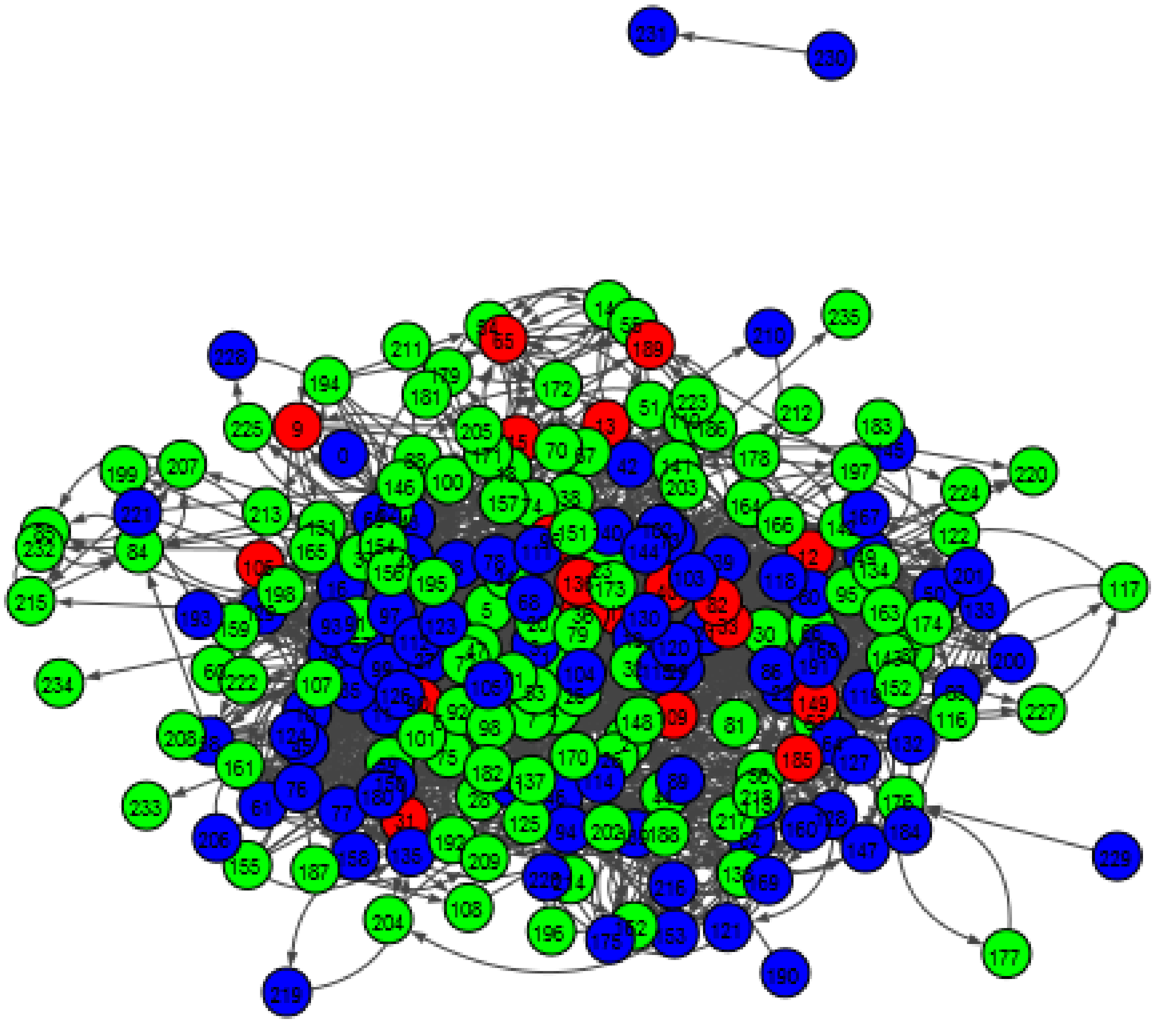

Figure 18: Twitter Social Network: Output CELF algorithm

5.5 YouTube Dataset

YouTube is a popular social network for sharing of videos. We use this dataset to construct a directed social network graph model. This dataset is obtained from the Github public open datasets [20]. As a part of the construction of social network from this data, the first step was the pre-processing of the data.

5.5.1 Data Pre-processing:

The raw dataset obtained is in the form of tab separated text files. It consisted of 60,205 records. Each entry in the dataset consisted of below fields.

video ID : Unique 11-digit string id

uploader : Username of the uploader

age : an integer number of days between the date when the video was uploaded and Feb.15, 2007

category : video category

length: length of the video

views : number of views for the video

rate : speed rate of the video

ratings : ratings received for the video

comments : number of the comments

related IDs : up to 20 related video IDs

To construct social graph, we use the videos whose length is greater than 2000 seconds. The videos form the vertices in the graph. A video is connected to all the videos in its related videos field by the edges. While pre-processing the data, a metadata file is created which lists the information of the videos. Hence, we can find the information like category, uploader's username, length of the video etc. of the resultant influencers seed set of videos.

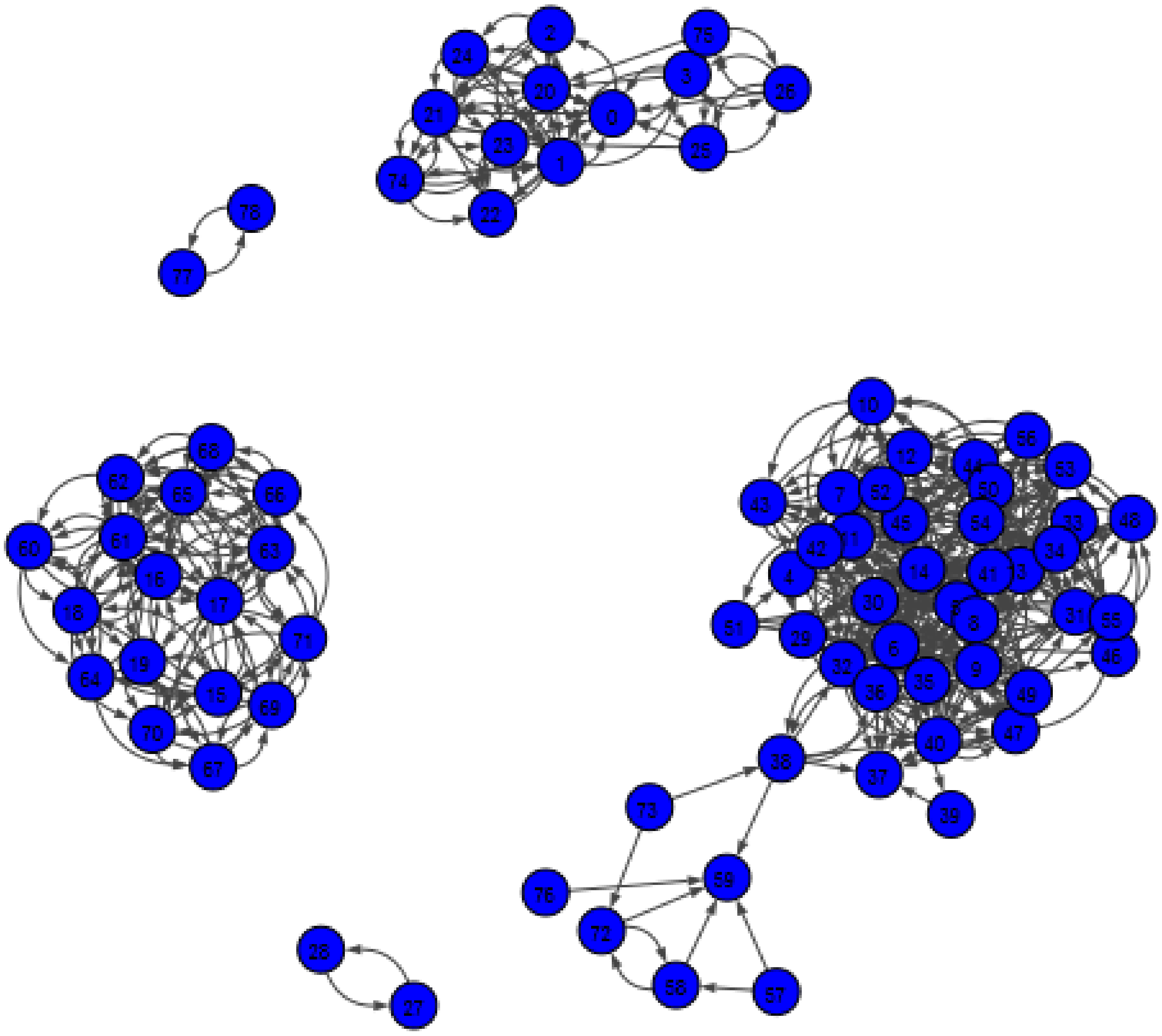
Figure 19: YouTube Social Network Graph

5.5.2 Results:

Figure 19 represents the social network graph obtained from this dataset. Here, the vertices represent the videos. A video is connected to its related videos by the edges. The graph consists of 79 vertices and 551 edges.

When we execute Algorithm 6, on the input graph to find influencers *seed set* of size 16 we get influencers as below. In Algorithm 6, we use vertex cover discovery as a preliminary step of the CELF algorithm.

Table 1: YouTube Social Network: Output influencer seed set by CELF with Vertex Cover

VideoName	Uploaded By	Category	Length
$hapkRYxCU_8$	GoogleDevelopers	Science and Technology	2913
kJGG1On2dIA	kobiyal	Entertainment	2465
$5jle1OJI_AQ$	kobiyal	Entertainment	3125
2iSsDlApXwk	kobiyal	Entertainment	2210
3-km-JgOMOk	kobiyal	Entertainment	2779
RhSB8ony23A	kobiyal	Entertainment	2801
O3flLCah7OU	kobiyal	Entertainment	2375
2zhcscszZtk	kobiyal	Entertainment	2442
8RIzPes-f4A	kobiyal	Entertainment	2566
$_OHOA9GCuZk$	kobiyal	Entertainment	3170
6fHMoi-IiHI	kobiyal	Entertainment	2407
iWGHF8hTWp4	kobiyal	Entertainment	4304
Qr1aD2frMMs	kobiyal	Entertainment	2293
QZz4ZDvfc1Y	kobiyal	Entertainment	2686
1dJMmVnF5L8	tokyomx	Entertainment	2685
mqH5RFWYj3M	tokyomx	Entertainment	2502

From Table 1, we see that the majority of the influential videos belong to the category *Entertainment*. Also, among the influential videos, maximum number of videos are uploaded by user *Kobiyal*.

We compute the influence spread of this *seed set* using Algorithm 3. The total number of vertices influenced by this *seed set* is 66.

Figure 20 shows the output of the *seed set* detection algorithm 6 and the spread estimation algorithm 3 on the YouTube social network. The vertices in red are the influencers whereas the vertices in green represent the number of vertices influenced by the *seed set* of influencers.

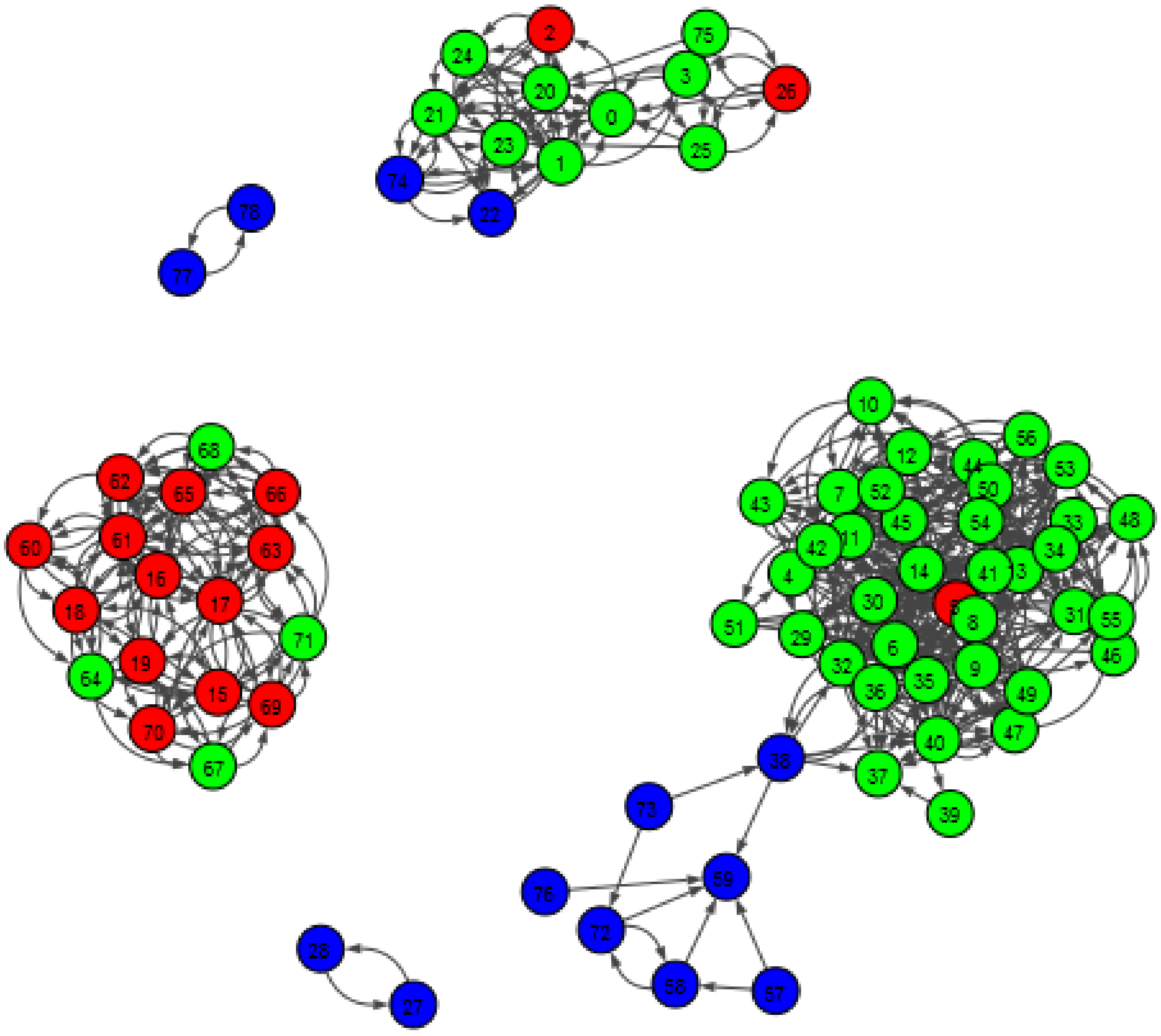

Figure 20: YouTube Social Network: Output CELF with Vertex Cover

We also find the seed set using Algorithm 5. Table 2 summarizes the information of influencer *seed set* obtained using Algorithm 5.

Table 2: YouTube Social Network: Output influencer seed set by CELF algorithm

VideoName	Uploaded By	Category	Length
$hapkRYxCU_8$	GoogleDevelopers	Science and Technology	2913
kJGG1On2dIA	kobiyal	Entertainment	2465
$5jle1OJI_AQ$	kobiyal	Entertainment	3125
2iSsDlApXwk	kobiyal	Entertainment	2210
NVllAEPHTn4	kobiyal	Entertainment	3350
yxP4BydUaaA	kobiyal	Entertainment	2671
3-km-JgOMOk	kobiyal	Entertainment	2779
6fHMoi-IiHI	kobiyal	Entertainment	2407
$_OHOA9GCuZk$	kobiyal	Entertainment	3170
Qr1aD2frMMs	kobiyal	Entertainment	2293
iWGHF8hTWp4	kobiyal	Entertainment	4304
RhSB8ony23A	kobiyal	Entertainment	2801
Ekv14HTwmBQ	kobiyal	Entertainment	2347
QZz4ZDvfc1Y	kobiyal	Entertainment	2686
2zhcscszZtk	kobiyal	Entertainment	2442
O3flLCah7OU	kobiyal	Entertainment	2375

Similar to the results in Table 1, from Table 2 we see that the majority of the influential videos belong to the category *Entertainment*. Also, among the influential videos, maximum number of videos are uploaded by user *Kobiyal*.

We compute the influence spread of this *seed set* using Algorithm 3. The total number of vertices influenced by this *seed set* is 55.

Figure 21 shows the output of the *seed set* detection Algorithm 5 and spread estimation Algorithm 3 on YouTube social network. The vertices in red are the influencers whereas the vertices in green represent the number of vertices influenced by the *seed set* of influencers.

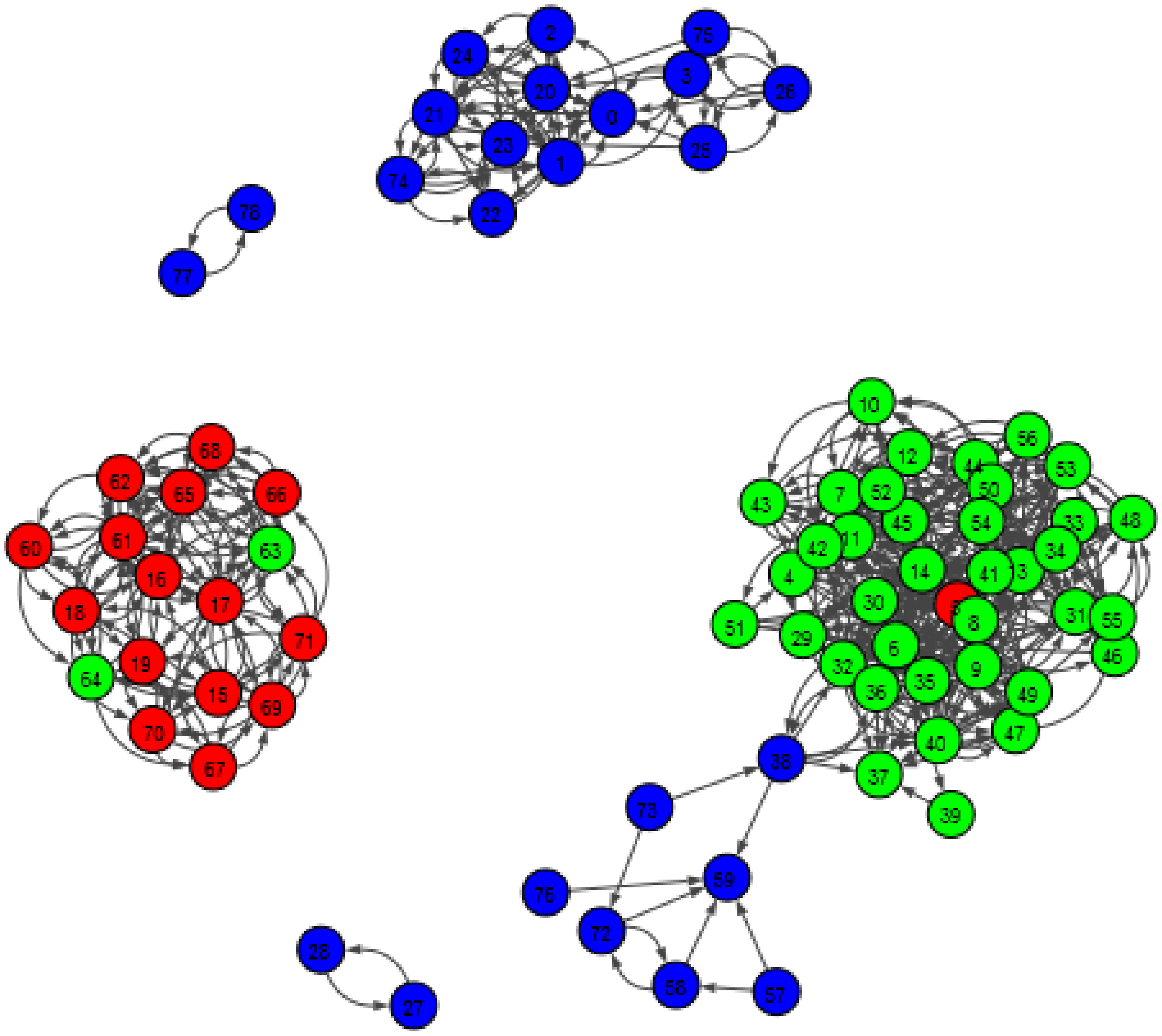

Figure 21: YouTube Social Network: Output CELF algorithm

Similarly, we compute the *seed set* of influencers using greedy Algorithm 2. Table 3 summarizes the information of the influencers obtained using Algorithm 2.

Table 3: YouTube Social Network: Influencer seed set by Greedy Algorithm

VideoName	Uploaded By	Category	Length
o5yrJ2irAmI	tokyomx	Entertainment	2552
sZYueEqL2-A	tokyomx	Entertainment	2648
mqH5RFWYj3M	tokyomx	Entertainment	2502
H2K9JDjh-vA	tokyomx	Entertainment	2842
-F-3E8pyjFo	GoogleDevelopers	Science and Technology	3465
$hapkRYxCU_8$	GoogleDevelopers	Science and Technology	2913
vwhdo-kMgrc	googletechtalks	People and Blogs	3145
chmANWyUfrs	googletechtalks	People and Blogs	3035
Wolw2gOARPk	soultimepromotions	Music	3627
2iSsDlApXwk	kobiyal	Entertainment	2210
EVXoWiMsQAU	tokyomx	Entertainment	2649
cdTVcFo2EQw	googletechtalks	People and Blogs	3048
1dJMmVnF5L8	tokyomx	Entertainment	2685
-alaG2pBCpk	uchannel	People and Blogs	3550
$5jle1OJI_AQ$	kobiyal	Entertainment	3125
kJGG1On2dIA	kobiyal	Entertainment	2465

From Table 3, we see that the majority of the videos have category Entertainment. However, the uploaders of these videos vary as opposed to the previous results we got from the CELF algorithms.

We compute the influence spread of this *seed set* using Linear the Algorithm 3. The total number of vertices influenced by this *seed set* is 79. Thus, we see that with greedy Algorithm 2 the *seed set* influences all the vertices in the graph.

Figure 22 shows the output of Algorithm 2. The vertices in red are the influencers whereas the vertices in green represent the influence spread of the *seed set*.

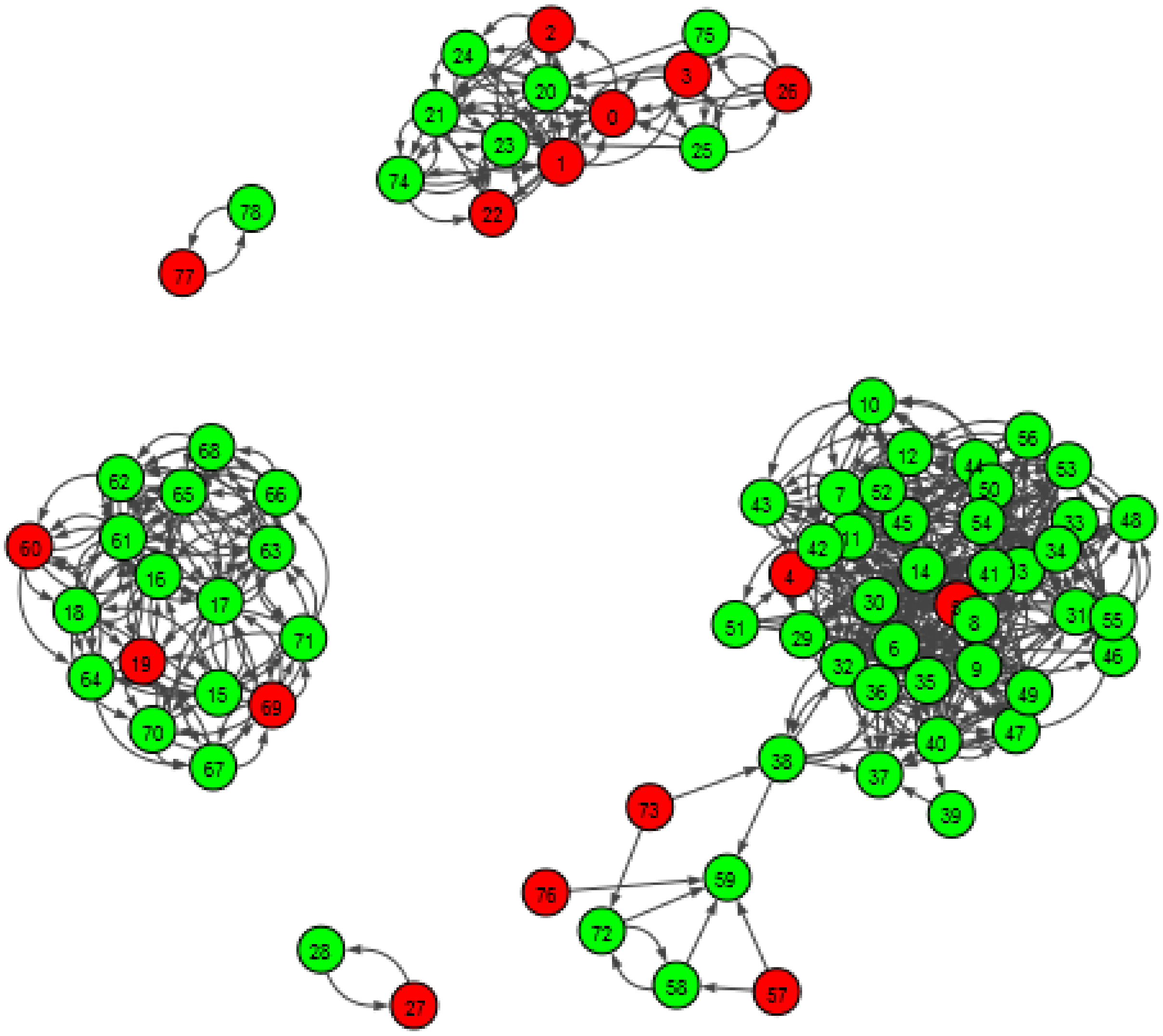

Figure 22: YouTube Social Network: Output Greedy algorithm

While performing the experiments, different values of *seed set* size were used to observe the performance of the algorithms. The performance of the algorithms is measured in terms of the number of vertices influenced by the *seed set*. The computation time for each *seed set* size run is compared for different algorithms. Table 5 summarizes the number of vertices influenced by each algorithm for different *seed set* sizes. In addition, Table 4 summarizes the computation time required by each algorithm for different *seed set* sizes.

Table 4: YouTube Social Network: Computation Time in seconds comparison

Seed Set Size (K)	CELF with Vertex Cover	CELF	Greedy algorithm
4	6.06	5.65	33.177
8	5.789	5.73	70.82
12	5.681	5.33	98.79
16	6.153	5.625	123.78
20	6.63	6.37	160.1
24	7.01	6.57	178.121
28	7.55	6.99	174.174

Table 5: YouTube Social Network: Spread Estimations comparison

Seed Set Size (K)	CELF with Vertex Cover	CELF	Greedy algorithm
4	55	55	64
8	55	55	73
12	55	55	79
16	66	55	79
20	68	66	79
24	70	68	79
28	74	72	79

Figure 23 shows a graphical comparison of the modified CELF Algorithm 6, CELF Algorithm 5 and greedy Algorithm 2 in terms of the spread estimations attained for different *seed set* sizes.

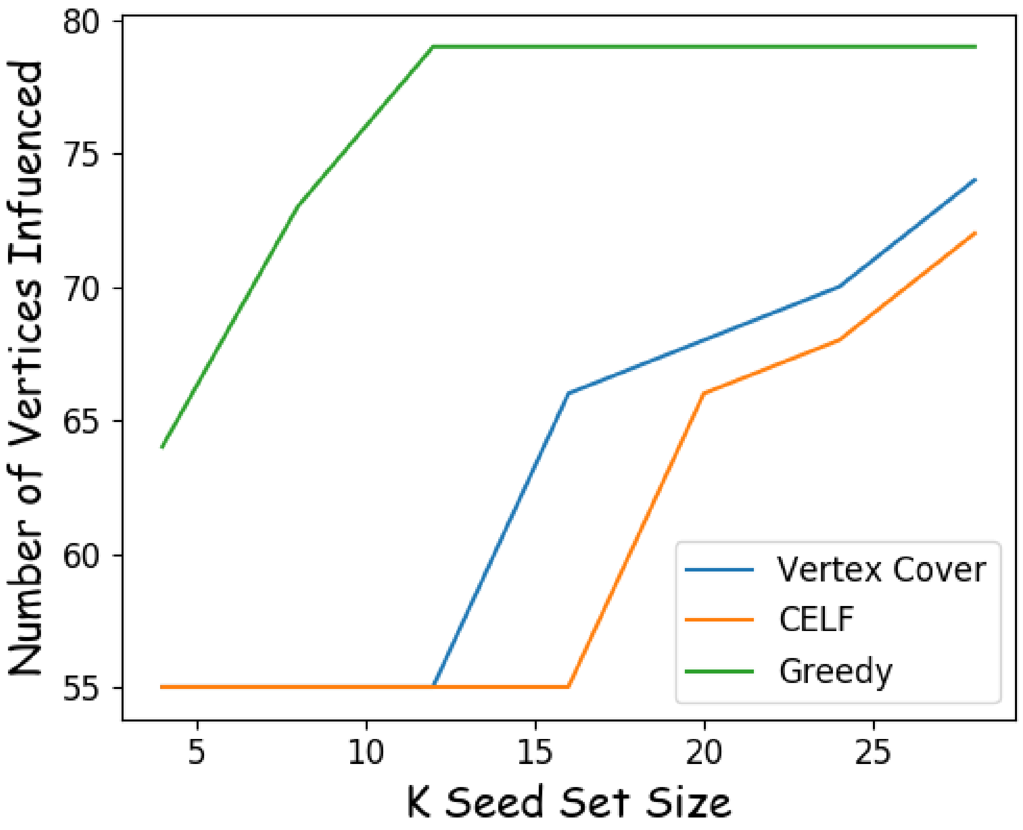

Figure 23: YouTube Social Network: Spread Estimations Comparison

We observe that *seed set* detected by Greedy Algorithm 2 attains maximum spread for different seed set sizes. The spread attained by the Algorithm 6 is better than that of Algorithm 5.

Figure 24 shows a graphical comparison of modified CELF Algorithm 6, CELF Algorithm 5 and greedy Algorithm 2 in terms of the computation times for different *seed set* sizes.

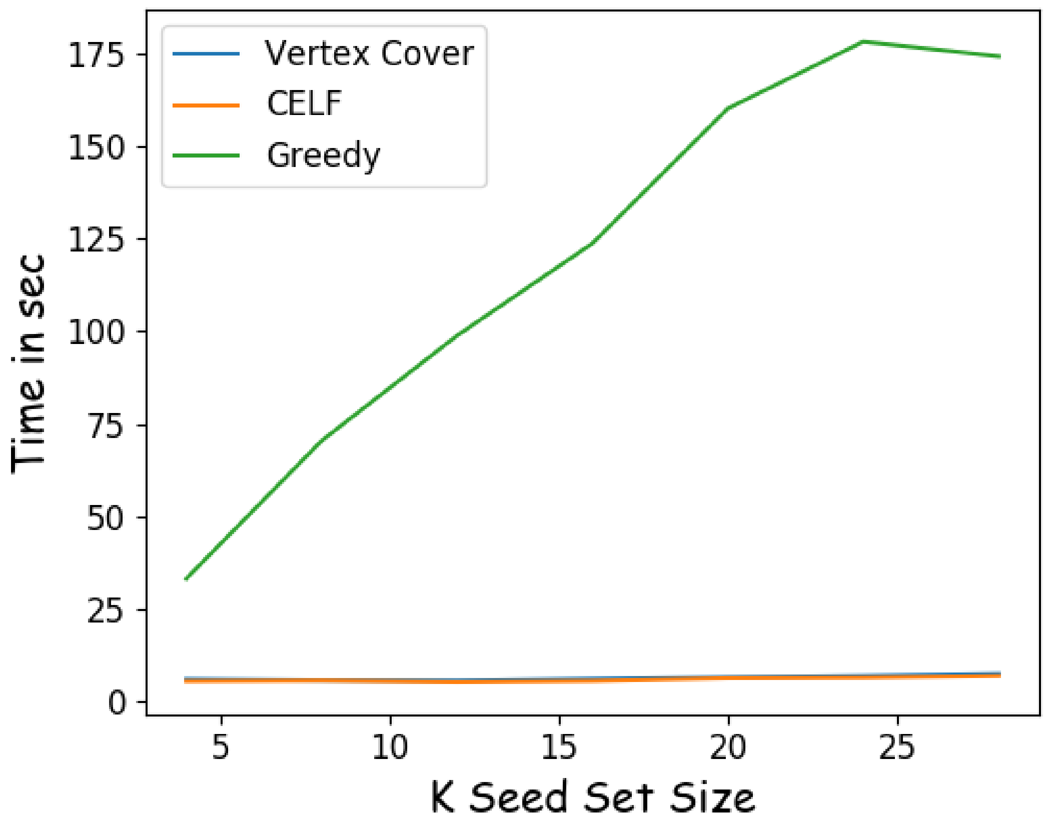

Figure 24: YouTube Social Network: Computation Times Comparison

We observe that the time taken by the Greedy Algorithm is very large as compared to the CELF algorithms. The computation time for Algorithm 6 and Algorithm 5 are comparable.

Thus, even though *seed set* detected by Greedy Algorithm 2 has better spread estimation, the computation time is very large as compared to CELF algorithms. Thus, Greedy Algorithm 2 is not suitable for large graphs.

5.6 Netscience Co-authorship Network

The Netscience dataset is a co-authorship network of scientists working on network theory and experiment, as compiled by M. Newman in May 2006 [15]. The network was compiled from the bibliographies of two review articles on networks, M. E. J. Newman, SIAM Review 45, 167-256 (2003) and S. Boccaletti et al., Physics Reports 424, 175-308 (2006), with a few additional references added by hand. This is an undirected network with authors forming the vertices of graph and the authors who co-authored a paper are connected by edges.
To detect the *seed set* of influencers we use Algorithm 4. Once the *seed set* is detected, we measure the number of vertices it influences by Algorithm 3.
The results obtained by Algorithm 4 are compared with the modified CELF Algorithm 6 which uses vertex cover computation as the preliminary step and the CELF Algorithm 5.
Figure 25 represents the social network graph for the Netscience co-authorship network. This graph has 1589 vertices and 2742 edges and is undirected in nature.

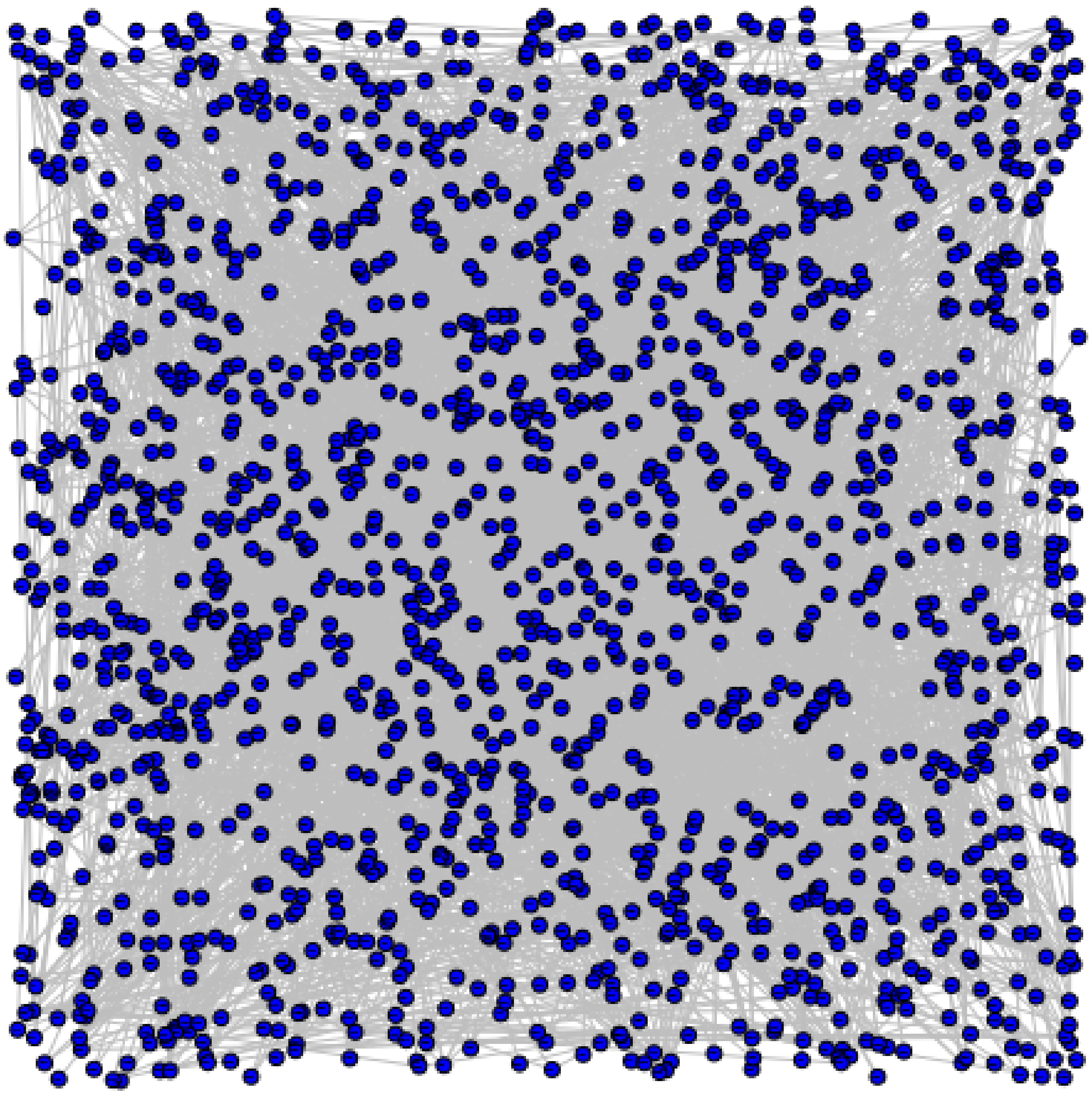

Figure 25: Netscience Co-authorship Social Network

Figure 26 represents the output social graph we get on executing the *seed set* detection Algorithm 4 for *seed set* size of 160 and the spread estimation Algorithm 3. The vertices in red are the influencers and the vertices in green represent the vertices influenced by the *seed set*. In total the *seed set* influences 379 vertices.

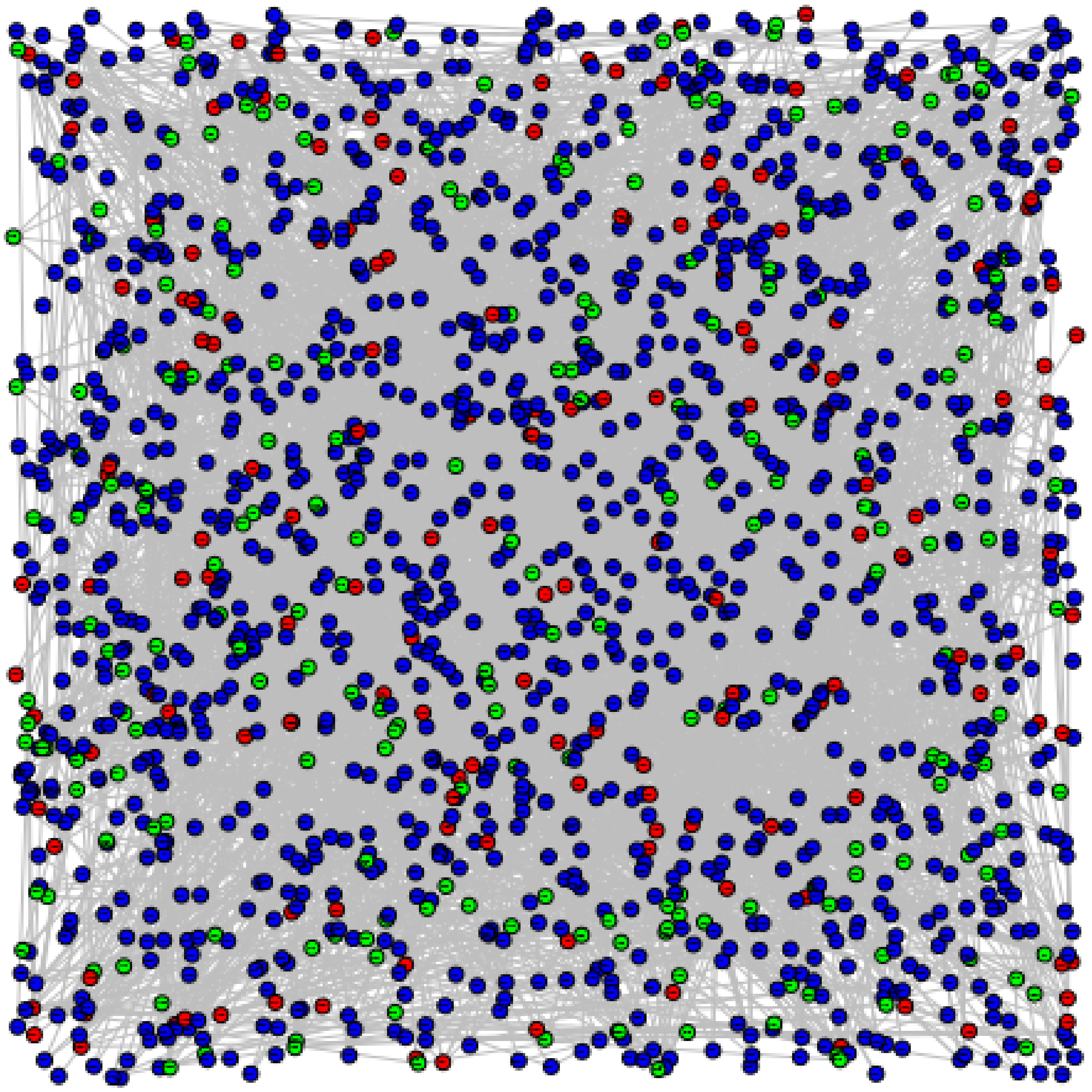

Figure 26: Netscience Co-authorship Network: Output graph for *seed set* detection using spanning graph

Figure 27 represents the output social graph we get on executing the *seed set* detection Algorithm 6 for *seed set* size of 160 and the spread estimation Algorithm 3. The vertices in red are the influencers and the vertices in green represent the vertices influenced by the *seed set*. In total the *seed set* influences 472 vertices.

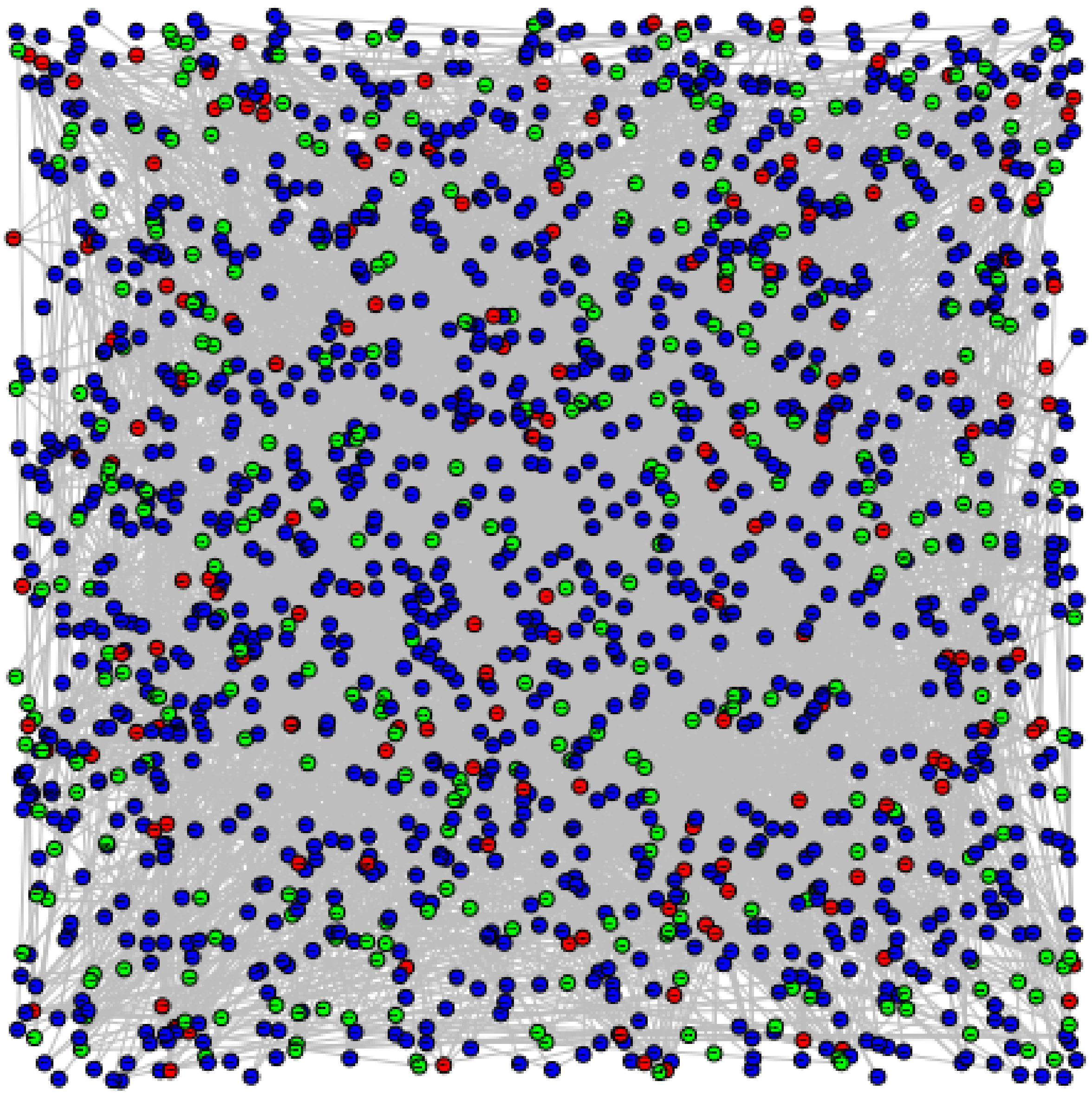

Figure 27: Netscience Co-authorship Network: Output graph for seed set detection using CELF with vertex cover

Figure 28 represents the output social graph we get on executing the *seed set* detection Algorithm 5 for *seed set* size of 160 and the spread estimation Algorithm 3. The vertices in red are the influencers and the vertices in green represent the vertices influenced by the *seed set*. In total the *seed set* influences 389 vertices.

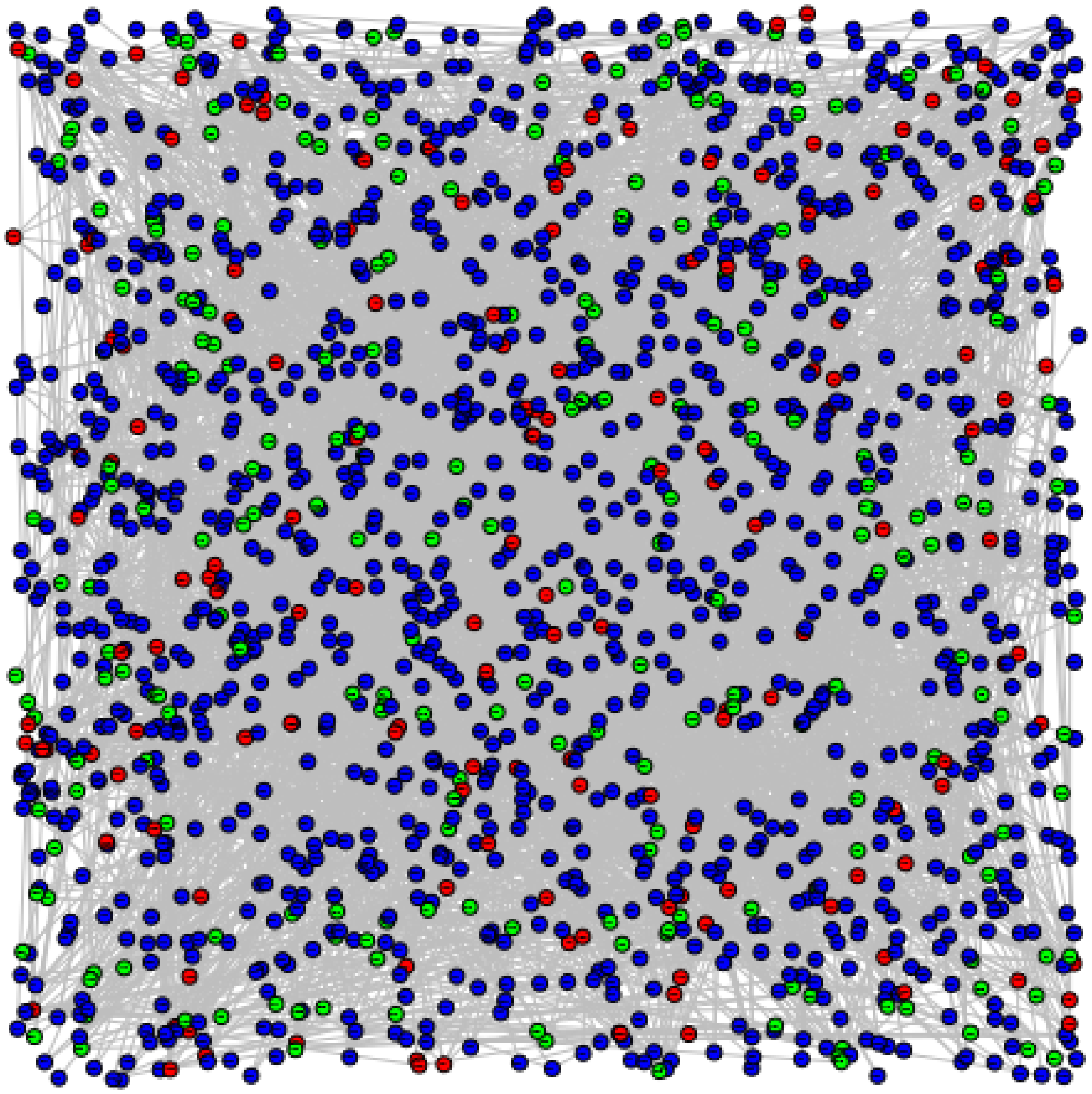
Figure 28: Netscience Co-authorship Network: Output graph for seed set detection using CELF

While carrying out the experiments, different values of *seed set* size are used to observe the performance of the algorithms. The performance of the algorithms is measured in terms of the number of vertices influenced by the *seed set*. The computation time for each *seed set* size run is compared for different algorithms. Table 6 summarizes the number of vertices influenced by each algorithm for different *seed set* sizes. In addition, Table 7 summarizes the computation time required by each algorithm for different *seed set* sizes.

Table 6: Netscience Co-authorship Network: Spread Estimations comparison

Seed Set Size (K)	Spanning Graph	CELF with Vertex Cover	CELF
160	379	472	389
320	379	674	540
480	587	895	681
640	791	1128	768
800	948	1332	959

Table 7: Netscience Co-authorship Network: Computation Time in seconds comparison

Seed Set Size (K)	Spanning Graph	CELF with Vertex Cover	CELF
160	7.51	936.743	2298.87
320	6.84	1534.34	3102.8
480	7.8	1714.86	4724.67
640	8.48	2176.94	3924.95
800	9.32	3103.36	5345.23

Figure 29 shows the graphical representation comparing the performance of the Algorithms 4, 5 and 6 in terms of the number of vertices influenced by the *seed set* obtained by running different sizes of *seed set*.

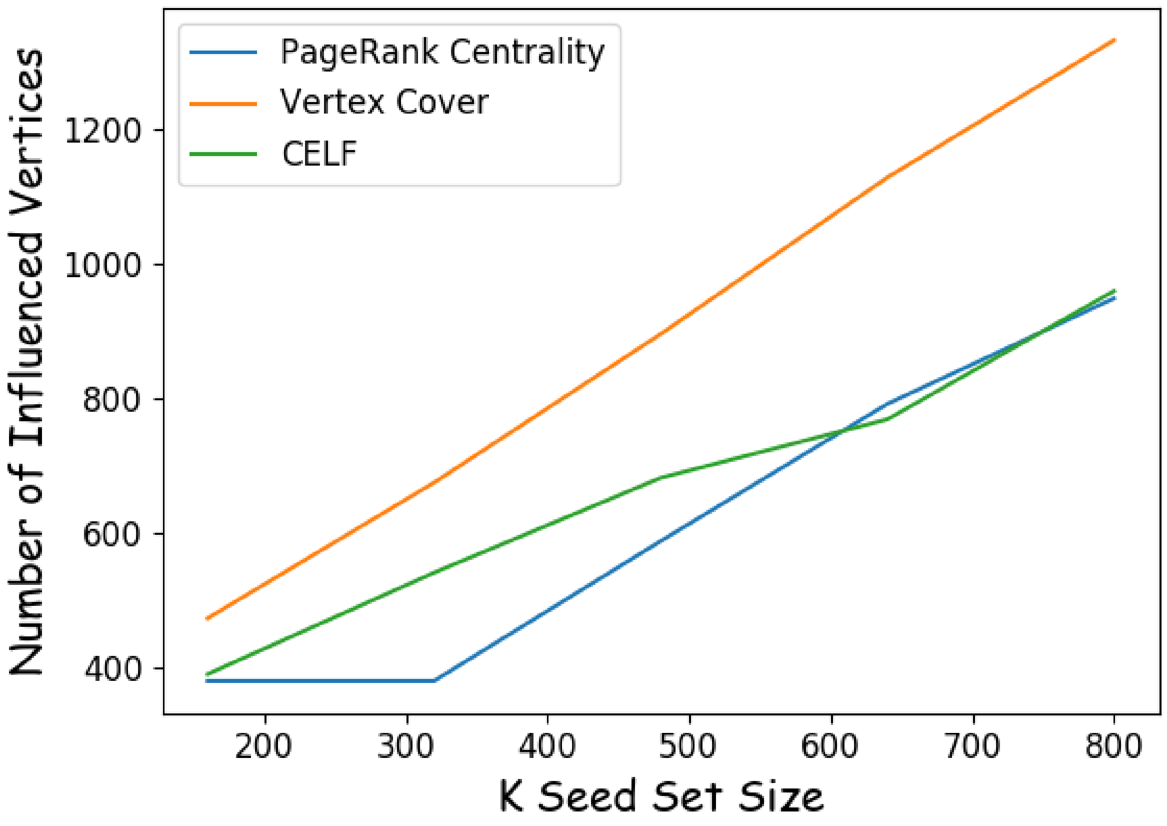

Figure 29: Netscience Co-authorship Network: Spread Estimations comparison

We observe that the influence spread of Algorithm 6 increases as the *seed set* size increases. The performance of Algorithm 6 in terms of the number of vertices influenced by particular *seed set* is better than the other two algorithms. The results of the other algorithms are comparable with each other.

Figure 30 shows the graphical representation comparing the performance of Algorithms 4, 5 and 6 in terms of the computation times. We carry out the experiments for different sizes of *seed set*.

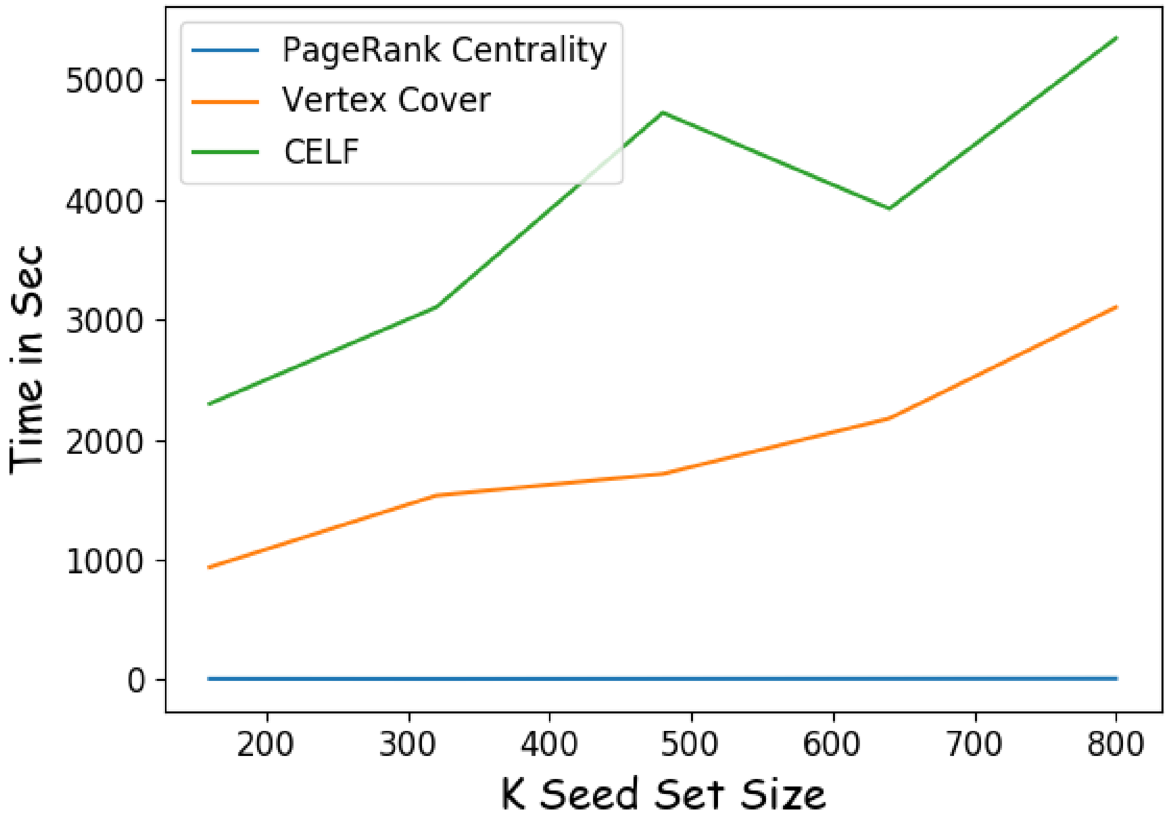

Figure 30: Netscience Co-authorship Network: Computation Time comparison

On observing the computation time of each algorithm from the graph in Figure 30, Algorithm 4 performs better than Algorithm 5. The computation time required by Algorithm 4 is less as compared to the other two. Also, it remains constant for increasing *seed set* size. For Algorithm 5, the computation time increases as the *seed set* size increases.

Thus, we observe that Algorithm 6 achieves larger influence spread but has large computation time. The influence spread of Algorithm 4 is slightly less as compared to Algorithm 6, however its computation time is much lesser as compared to CELF based algorithms.

In this chapter, we discussed about the experiments performed as a part of this project. In chapter 6, we discuss some of the applications of *influence maximization* problem in real world.

CHAPTER 6

Application

In this chapter, we discuss about the real world applications of *influence maximization* problem. The section 6.1 discusses the use of *influence maximization* in the operations of *Smart Cities*.

6.1 Influence Maximization in Smart Cities

Social network platforms have plethora of applications. Use of social networks in building of smart cities is one such application [10]. Building a smart city is a gradual process, where identifying current problems that a city faces is the primary step. Examining the different factors and players of each local government can be the first step towards proposing smart city initiatives. Citizens play an important role in identifying such challenges. Involvement of the public is important to build a smart city. People must voice their opinion on various issues faced as well as share ideas for the betterment of the city with others. Social networks is widely used as a communication platform to share information. Various social networking platforms like Facebook, Twitter etc provide a central location for sharing information among remote users. The like, comment and forming of groups features of Facebook or re-tweet, follow features of Twitter provide an effective way of sharing information. Micro blogging is another popular social medium where users express their opinions about any issue with others. Use of these social networks will benefit in building the smart cities, in the planning and managing of resources. People express different opinions about different issues using social networks. We can group these people in the network, based on the common issues as well as common sentiments about a particular issue to form different communities. The communities, can propose solutions and ideas for overcoming the problems or challenges faced using technology. Polls and surveys can be conducted via social medias, to finalize such proposals. People can also keep track of the status of the implementation of plan via social networks. Thus, use of social networks increases the public participation in smart cities and enhances local government accountability.

In each community, some people are more active in voicing their opinions. For example, some micro bloggers are more popular and have large number of followers, some journalists are more effective in spreading the information. They act as influencers in their network. The opinion and behavior of the rest of the people participating in that network is influenced by such influencers. Their positive comments and writing motivates people. Finding such influential individuals, called influencers, in a community will influence the community at large. This will benefit in campaigns of spreading awareness, increasing public participation in the planning process etc. Thus, finding influencers in communities can lead to effective and organized planning to build smart cities. A city can be modeled into a social graph. We can use the proposed Algorithm 4, for the undirected graph to find *k-influencers seed set* in each community, where each community is considered a subgraph. Similarly, for directed graphs, we can use Algorithm 6 *k-influencers seed set* detection. The information spread by this subset of people across the community can be measured using Algorithm 3.

Some of the other applications of *influence maximization* are viral marketing, finding the influential blogger, identifying cause of contaminants, finding influential person on social network platforms like LinkedIn and Twitter, outbreak detection and optimal placement of sensors.

CHAPTER 7

Conclusion

Social networks are widely used as a communication platform. *Influence maximization* problem, is a popular problem in social networks and has many applications like viral marketing, building smart cities, detection of contaminants, placing sensors etc. As a part of this project, I studied approximation based solutions for the *influence maximization problem.* The greedy approximation algorithm, has an efficiency of at least 63%. Moreover, the greedy algorithm selects one vertex at a time and computes its influence spread, which is time consuming and not suitable for large graphs. CELF algorithm, an improvisation of the greedy algorithm, reduces the spread estimation calls by maintaining a priority queue of the vertices in the descending order of the marginal gain. It performs 700 times faster than the simple greedy approximation algorithm. As a part of this project, I modified the CELF algorithm to add a vertex cover discovery as a preliminary step of the algorithm. The CELF algorithm, in the preliminary step computes the marginal gain for all the vertices. Hence, the discovery initially of a vertex cover improves the computation time of the CELF algorithm since it considers only those vertices.

For undirected graphs, we use an acyclic spanning graph based approach for *seed set* detection of *k-influencers.* This algorithm uses PageRank centrality to construct the acyclic spanning graph. From the experiments, we observe that the computation time of the acyclic spanning graph based approach is better than the greedy approximation algorithm with comparable spread. Also, the computation time of the spanning graph based algorithm remains constant with an increase of *seed set* size whereas the computation time of greedy algorithm increases as the seed set size increases. Once the *seed set* is detected we compute the spread of the influence using the Linear Threshold model. We use the real world social networks and synthetic graphs to perform the experiments.

As a part of the future work, I plan to explore probability based Independent Cascade model for influence spread estimation. In addition, the implementation framework, will be enhanced to support the distributed graph processing.

Bibliography

[1] Ali Sajedi Badashian and Eleni Stroulia. “Measuring user influence in GitHub: the million follower fallacy”. In: *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*. ACM. 2016, pp. 15–21.

[2] Albert-László Barabási and Réka Albert. “Emergence of scaling in random networks”. In: *science* 286.5439 (1999), pp. 509–512.

[3] Wei Chen, Yajun Wang, and Siyu Yang. “Efficient influence maximization in social networks”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 199–208.

[4] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[5] Magdalini Eirinaki, Nuno Moniz, and Katerina Potika. “Threshold-Bounded Influence Dominating Sets for Recommendations in Social Networks”. In: ().

[6] Ibrahima Gaye et al. “Spanning graph for maximizing the influence spread in Social Networks”. In: *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*. 2015, pp. 1389–1394. DOI: `10.1145/2808797.2809309`. URL: `http://doi.acm.org/10.1145/2808797.2809309`.

[7] Michelle Girvan and Mark EJ Newman. “Community structure in social and biological networks”. In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[8] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. “SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model”. In: *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*. 2011, pp. 211–220. DOI: 10.1109/ICDM.2011.132. URL: http://dx.doi.org/10.1109/ICDM.2011.132.

[9] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. “Celf++: optimizing the greedy algorithm for influence maximization in social networks”. In: *Proceedings of the 20th international conference companion on World wide web*. ACM. 2011, pp. 47–48.

[10] Madhura Kaple, Ketki Kulkarni, and Katerina Potika. “Viral Marketing for Smart Cities: Influencers in Social Network Communities”. In: *9th IEEE International Workshop on Big Data Appications in Smart City Development*. 2017.

[11] David Kempe, Jon M. Kleinberg, and Éva Tardos. “Maximizing the Spread of Influence through a Social Network”. In: *Theory of Computing* 11 (2015), pp. 105–147. DOI: 10.4086/toc.2015.v011a004. URL: http://dx.doi.org/10.4086/toc.2015.v011a004.

[12] Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. Vol. 37. Addison-Wesley Reading, 1993.

[13] Jure Leskovec et al. “Cost-effective outbreak detection in networks”. In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*. 2007, pp. 420–429. DOI: 10.1145/1281192.1281239. URL: http://doi.acm.org/10.1145/1281192.1281239.

[14] Ashish Nargundkar and YS Rao. "InfluenceRank: A machine learning approach to measure influence of Twitter users". In: *Recent Trends in Information Technology (ICRTIT), 2016 International Conference on*. IEEE. 2016, pp. 1–6.

[15] Mark EJ Newman. "Finding community structure in networks using the eigenvectors of matrices". In: *Physical review E* 74.3 (2006), p. 036104.

[16] *Python-igraph*. URL: `http://igraph.org/python/` (visited on 04/10/2017).

[17] Gerasimos Razis and Ioannis Anagnostopoulos. "InfluenceTracker: Rating the Impact of a Twitter Account". In: *Artificial Intelligence Applications and Innovations - AIAI 2014 Workshops: CoPA, MHDW, IIVC, and MT4BD, Rhodes, Greece, September 19-21, 2014. Proceedings*. 2014, pp. 184–195. DOI: `10.1007/978-3-662-44722-2_20`.

[18] Daniel M Romero et al. "Influence and passivity in social media". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 18–33.

[19] *SNAP dataset (Twitter)*. URL: `https://snap.stanford.edu/data/egonets-Twitter.html` (visited on 04/30/2017).

[20] *YouTube Social Network Dataset*. URL: `https://github.com/caesar0301/awesome-public-datasets#social-networks` (visited on 04/10/2017).

[21] *Zachary's Karate club*. URL: `https://networkdata.ics.uci.edu/data.php?id=105` (visited on 04/10/2017).

[22] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social media mining: an introduction*. Cambridge University Press, 2014.