2012

Discovering Influential Nodes from Social Trust Network

Sabbir Ahmed
University of Windsor

Discovering Influential Nodes from Social Trust Network

By

Sabbir Ahmed

A Thesis
Submitted to the Faculty of Graduate Studies through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2012

Discovering Influential Nodes from Social Trust Network

By

Sabbir Ahmed

APPROVED BY:

Dr. Eugene H. Kim, External Reader

Department of Physics

Dr. Alioune Ngom, Internal Reader

School of Computer Science

Dr. Christie I. Ezeife, Advisor

School of Computer Science

Dr. Richard Frost, Chair

School of Computer Science

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

The goal of viral marketing is that, by the virtue of mouth to mouth word spread, a small set of *influential* customers can influence greater number of customers. Influence maximization (IM) task is to discover such influential nodes (or customers) from a social network. Existing algorithms adopt Greedy based approaches, which assume only positive influence among users. But in real life network, such as trust network, one can also get *negatively* influenced.

In this research we propose a model, called T-GT model, considering both positive and negative influence. To solve IM under this model, a trust network where relationships among users are either 'trust' or 'distrust' is considered. We first compute positive and negative influence by mining frequent patterns of actions performed. Then using local search a new algorithm, called MineSeedLS, is proposed. Experimental results on real trust network shows that our approach outperforms Greedy based approach by almost 35%.

KEYWORDS

Influence Maximization, Trust Network, Social Network, Data Mining.

DEDICATION

To my loving wife Tanya and my beautiful daughter Alaida.

ACKNOWLEDGEMENT

My sincere appreciation goes to my parents, wife and siblings. Your perseverance and words of encouragement gave me the extra energy to see this work through.

I will be an ingrate without recognising the invaluable tutoring and supervision from Dr. Christie Ezeife. Your constructive criticism and advice at all times gave me the needed drive to successfully complete this work. The research assistantship positions helped as well!!

Special thanks go to my external reader, Dr. Eugene H. Kim, my internal reader, Dr. Alioune Ngom for accepting to be in my thesis committee. Your decision, despite your tight schedules, to help in reading the thesis and providing valuable input is highly appreciated.

And lastly to friends and colleagues at University of Windsor, I say a very big thank you for your advice and support throughout the duration of this work.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Social Network Analysis

A social network is a social structure made up of individuals or entities (e.g. organization) also called "nodes", which are inter connected by various types of relationships, such as friendship, trust etc. Social Network Analysis (SNA) concentrates on techniques to analyze these relationships and information flows, between *nodes* in a social network, and produce formal models which facilitates understanding of the structure of a network as well as which network structure is more likely to emerge (Wellman and Berkowitz, 1988).

Popularity of online social networking sites (e.g. Facebook, Google+), caused a rise of social network data of very large scale. Researchers are actively involved in studying and understanding properties and structures of these networks and the challenges they pose by applying various data mining and machine learning methods to these data, such as Kempe et al (2003) and Leskovec et al. (2010). These studies are to better understand the online social structure, its growth and user behaviour etc. (Backstorm et al. 2010). Such studies can help developers of social network sites to improve user experience, make it scalable, and of course, profitable. Also tools and techniques for analysing and mining social network have various range of use in business processes such as marketing, sales etc. (Bonchi et al. 2011).

There are several examples of such works, for example Link Prediction, the task of predicting a future link (such as friendship) between two individual (nodes) in a social network. Link prediction is a key tool for friend recommendation system in many social network sites. Another example is the task of *influence maximization*, which is the problem of detecting a small subset of social network graph that could maximize the spread of influence (Kempe et al. 2003). Here ‘influence’ can be for a piece of information from government that needs to be spread to maximum possible members of a

social network, or it can be a new technology, such as new android phone, a company wants to promote etc. Kempe et al. (2003) defines the influence maximization problem as a *submodular function maximization* problem and provide greedy based solution for it.

Existing algorithms for influence maximization, such as Greedy (Kempe et al. 2003) and 'Lazy Forward' (Leskovec et al. 2006) based algorithms, requires that the influence probability, the probability of an individual adopting a product under the influence of another user, is known and given to the algorithm as input along with a social network graph. A social network graph can be easily constructed if the relationship (among users in a social network) data is explicitly available. However influence probabilities are not explicitly available (Goyal et al. 2008). In most of the literature reviewed influence probabilities are assumed and given as input.

To tackle this, researchers are recently looking into data mining techniques, such as frequent pattern mining (Goyal et al. 2008), to mine influence from user's action log. These techniques in general takes Action Log, which is a relation Actions (User, Action, Time), along with a social graph $G(V, E)$. Action log are extracted from log of user activities, such as rating a movie, in a social network site databases. Action log table contains tuples, for e.g. (u, a, t), which indicates that user u (such that $u \in V$) performed action a, at time t. Recent researches show that such action log can provide traces of *influence* among users in a social network (Goyal et al. 2011). In other words, if a user v rate "Mission Impossible" movie and later v's friend u does the same, then the action of rating the movie "Mission Impossible" propagates from user v to user u. This is to be noted that these works primarily consider only the *positive* influence among users. However a user can also get *negatively* influenced by another user, especially by a *distrusted* user. For example let us say Tom trust Mary and John, but do not trust Rob. Current solutions for IM will only consider influence (positive) of Mary and John on Tom and use it to compute the probability of Tom performing a certain task or action. Furthermore, in current models Tom's probability of performing an action (or adopting a product) will increase as the number of his friends performing the same action increases. However, we argue that, Tom's probability of performing an action (e.g. Buy iPhobe 4S) should also *decrease* if Tom's distrusted neighbours, such as Rob, also buy iPhobe 4S. And thus we should also consider *negative* influence in Influence Maximization tasks.

In this research a trust network is considered, which is a social network where we have both positive (e.g. friendship) and negative (e.g. foes) types of links or edges, to solve influence maximization where both positive and negative influence exist. We propose a new model; called Trust-General Threshold (TGT) Model, in which the probability of a user to adopt a product relies on both positive and negative influence probabilities. By extracting patterns of actions performed by users we present a pattern mining based method to compute the influence probabilities (both positive and negative), from Action Log using Bernoulli distribution. Furthermore we show that, influence maximization under the new TGT model cannot be solved with good approximation guarantee using existing methods, such as 'Lazy Forward' of Leskovec et al. (2006). This is mainly because existing works assume that probability of a user performing an action increases as more of its neighbours perform the same action. However in our model this is not that case as the probability may in fact decrease if its neighbours who perform the actions are not the trusted ones. Using local search techniques we propose a new algorithm, MineSeedLS, to solve IM under TGT model. We conduct experiments using dataset collected from Epinions.com and Wikipedia.com to evaluate our approach.

In remaining of chapter 1 we provide a brief background on data mining, social network and its analysis. Here we also discuss submodular function optimization as influence maximization problem can be formulated as this problem. Then we discuss the influence maximization problem and also provide problem statement for the thesis. In chapter 2 we provide a detailed related work on influence maximization and also discuss limitations of these works and motivation for the thesis. In chapter 3 we provide a proposed solution framework to solve influence maximization in trust network. In chapter 4 we provide various experimental results including comparisons between the existing and the proposed influence maximization techniques. Finally, Chapter 5 provides some concluding remarks.

1.2 Data mining

Data mining (also refer to as knowledge discovery from data or KDD) is the process of analysing data from different perspective and presenting it into meaningful information that can be used to make important and critical decisions. Agrawal and Srikant (1996) define data mining as a way of efficiently discovering interesting rules from large databases. This area of study is motivated by the need for solutions to decision support problems faced by organisations such as Banks, Retail stores etc.

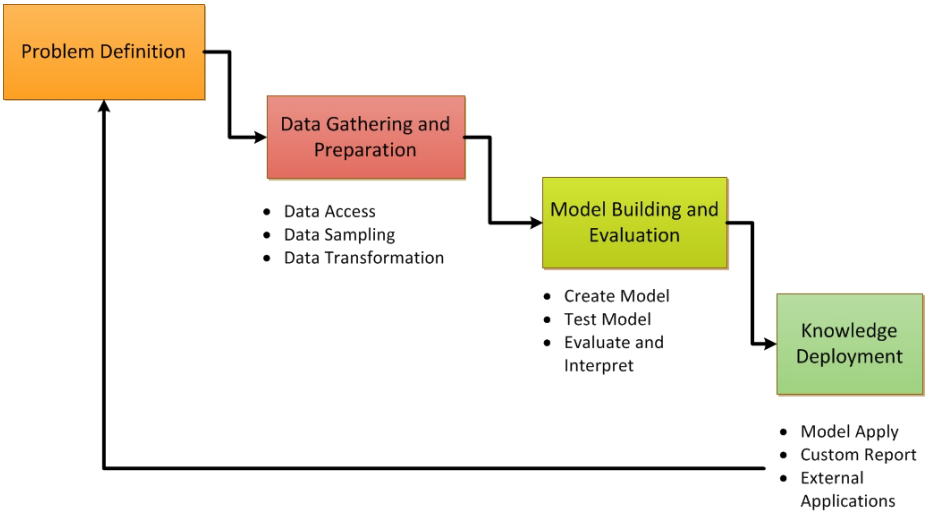


Figure 1: Data mining process. (http://docs.oracle.com)

Any data mining task first involve defining the problem or business goal. For example a business may want to find potential customers to target to initiate a viral marketing campaign. The next step is to gather and process available data. This typically involves creating databases or warehouses where data are then read using various techniques. Then the next process is to build models to analyse and forecast on future events. The entire data mining steps can be illustrated as shown in figure 1. Data mining algorithms typically employs techniques from statistics; machine learning and pattern recognition to search large databases automatically. One key factor that is important in this field is to ensure that information or knowledge extracted is accurate and conclusions drawn from data can be good enough to represent the general situation. Therefore it is also important to ensure that proper cleaning is done on the data so that so that the results generated by

the mining task are accurate. The process of discovering interesting information from large set of data often employs different techniques and approaches. Some of the approaches include:

- **Classification** – it is the task of assigning or *classifying* objects to one of several predefined categories or *class*. For example we may want to predict whether it is suitable to play tennis on a given day by looking into playing condition data from previous days. A collection of records, also known as *training set,* with their class labels is used as input data for a classification task. An example of an data of tennis play which can be used as training set for classification is in table 1 below:

Table 1: An example of a training set.

Temp	Outlook	Wind	Humidity	Play?
Hot	sunny	weak	High	no
Hot	sunny	strong	High	no
Hot	overcast	weak	High	yes
Mild	rain	weak	High	yes
Cool	rain	weak	Normal	yes
Cool	rain	strong	Normal	no
Cool	overcast	strong	Normal	yes
Mild	overcast	weak	High	no
Cool	sunny	weak	Normal	yes
Cool	rain	weak	Normal	yes
Mild	sunny	strong	Normal	yes
Mild	overcast	strong	High	yes
Hot	overcast	weak	Normal	yes
Mild	rain	strong	High	no

Attributes *outlook, temp, humid* and *wind* in the training set are called *independent* attributes and the attribute whose class we need to predict, i.e. '*play?'*

is called *dependent* attribute. The goal of any classification algorithm is to take training set data as input and produce a classification model which is then used to classify a new record of which the class or value for dependant attribute is unknown. Examples include decision tree classifiers, neural networks, naïve Bayes classifier etc. Figure 2 is a decision tree generated using the training set in table 1 above. A decision tree is a tree representation which is a collection of classification rules, one for each leaf node. In order to predict or classify an unknown record, the attribute values of the record are compared against the decision tree. A path from root of the decision tree to a leaf node which holds the predicted class of that record is traced.

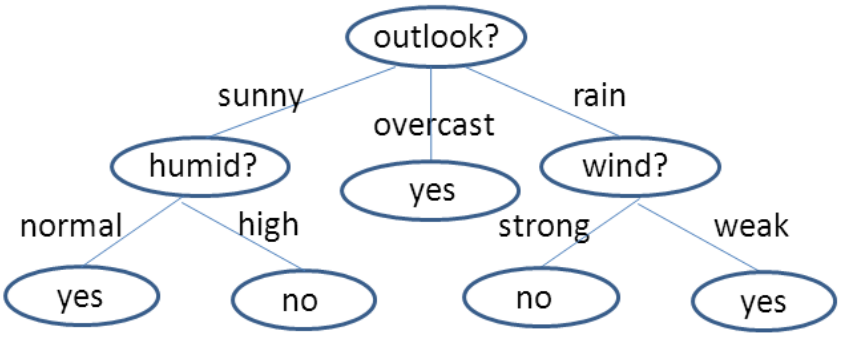


Figure 2: An example of a decision tree.

The decision tree is then used to predict the value of the target attribute '*play?'* for a new instance. For example, from the above decision tree model, if outlook is 'sunny' and humidity is 'high' we can predict 'no' for '*play?'* attribute according to the decision tree.

- **Clustering** – tasks seeks to discover records that are closely related and put them into different groups, or *clusters,* such that records in same cluster are more similar to each other than records that belong to other clusters. For example a business can look into their customer data and perform clustering analysis to segment customers into small groups for various marketing activities. K-means

algorithm is an example of a clustering algorithm. Figure 3 below shows a simple example of clusters of 3 groups separated based on debt and income.

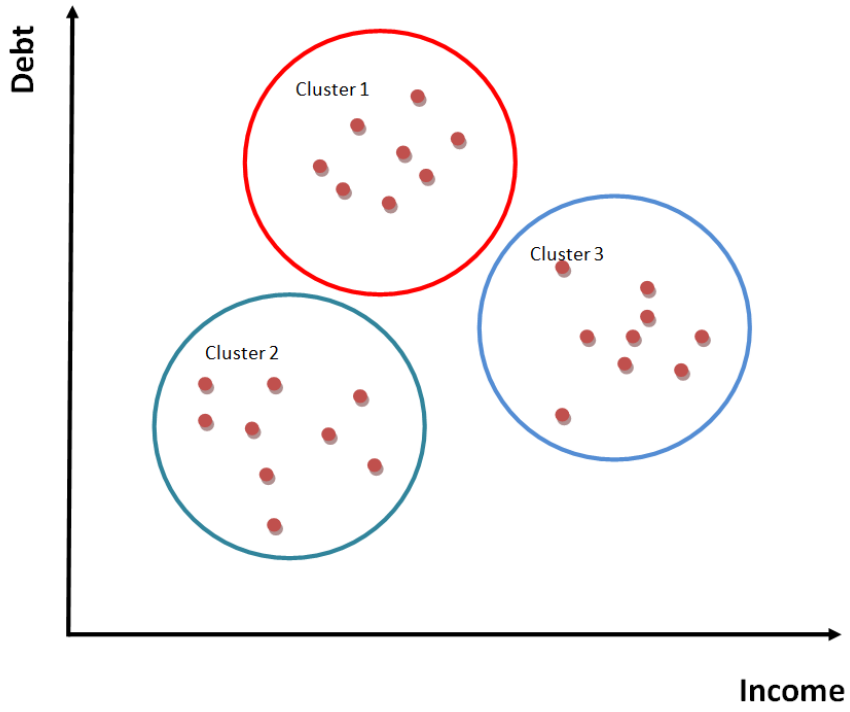


Figure 3: A clustering example.

- **Association rule mining** – is used to discover patterns that describe strongly associated features in data. It is often very useful to discover such interesting patterns or relationship hidden in large data sets. The uncovered relationships are represented in the form of *association rules* or set of frequent items (Agrawal et al. 1993). For example retail stores are interested in analysing the point of sales data to learn about purchasing behaviours of their customer. If they see, for example, 80% of their user buy *bagels* when they buy *cream cheese*, the retail store may put these items close to each other to encourage more sales. Association rule mining typically involves exploring transaction table (example Table 2). Each row in transaction table corresponds to a list of items bought by a customer. For example transaction 1 (TID 1) corresponds to a sale where the customer bought

bread and milk. The goal is to generate all possible patterns from the database, calculating their support (how often does the rule apply) and confidence (how often is the rule correct). Rules are generally simple, example, "If bread is purchased, then milk is purchased 60% of the time and this pattern occurs in 60% of all shopping baskets."

Table 2: Example of transaction table

TID	Items
1	{bread, milk}
2	{bread, diapers, beer, eggs}
3	{bread, diapers, beer, cola}
4	{bread, milk, diapers, beer}
5	{bread, milk, diapers, cola}

1.3 Social Network Graph and Properties

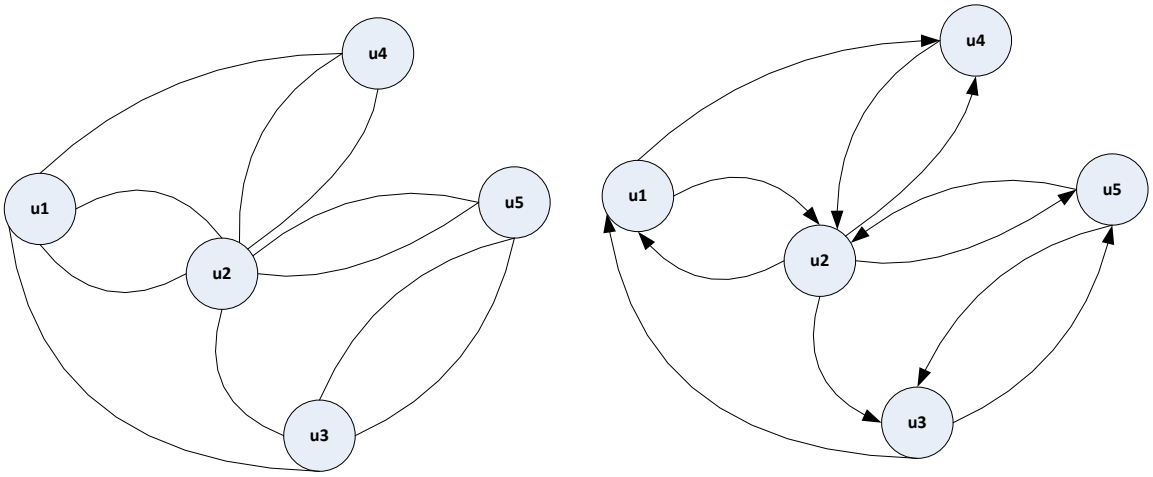
Graphs are used to specify relationships among a collection of items or nodes. It consists of a set of objects, called nodes, with certain pairs of these objects connected by links called edges. Graphs appear in many domains and context, whenever it is useful to model how things (or nodes) are either physically or logically connected to each other in a network structure (Easley and Kleinberg, 2010).

Following is the more formal definition of graph:

Definition 1 ***Graph*** - Graph G is a pair (V,E), where V is a set of vertices (or Nodes), and E is a set of edges between the vertices $E \subseteq \{(u,v) \mid u, v \in V\}$

The structures of social networks, most commonly, are modeled as directed or undirected graph $G(V, E)$. Where V is the set of all nodes in the network and E is the set of edges between nodes. In Figure 4(a), the relationship between the two ends of an edge is

symmetric (e.g. friendship); the edge simply connects them to each other. However in various other settings, it may be required to express *asymmetric* relationships. For example a node can point to another node but not vice versa. Such relationships in a social network are modeled as *directed* graph which consist of a set of nodes with a set of directed edges; that is each directed edge is a link from one node to another. For example in Email network we need to specify sender and receiver. An example of a directed graph is shown in figure 4(b), with edges represented by arrows.



a) Undirected graph *b) Directed graph*

Figure 4 – Example of Directed and Undirected Graph.

Graph is a popular way to model the interaction among the people in group or community. For example, in a friendship network, each vertex *V* corresponds to a person in that network, and an edge *E* between two such vertices represents an association of friendship among them. Each edge in a graph structure can also have weights assigned to them. Such graph models are known as weighted graphs, and are used to model structures in which each edge between a pair of nodes has some numerical value. For example, in an Email network graph, the weights of its edges may represent the number of emails sent from one node to another. By modeling of social network as graph and analyzing it, useful perspectives on range of social computing applications, such as friend recommendation system can be provided. Graph modeling of social network are also useful because they provide us with mathematical models of the network structures

(Easley and Kleinberg 2010). Let us consider the following social network data tables. Table 3 consist of list of individuals in a social network and Table 4 reports friendship relationship among these individuals.

Table 3: Sample table with user information

Userid	Name	Age	Sex	Location
101	Bob	32	M	Toronto
102	Mary	22	F	Windsor
201	Tanya	32	F	Dhaka
301	Ahmed	22	M	NewYork

Table 4: Relationship information of users in Table 3

Userid	Friend_Of	DateCreated
101	301	12-Mar-2007
301	201	22-Apr-2009
101	102	05-Jun-2011
102	301	02-Dec-2010

Based on these data we can model a simple social network graph as shown in Figure 5.

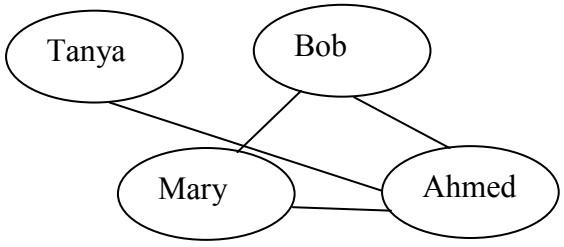


Figure 5 - Graph model of social network data in Table 3 and 4.

In the above graph $G(V,E)$, V is the set of individuals (or vertices) in the social network , i.e. V={Tanya, Bob, Ahmed, Mary}. And E is the set of all friendship links (or edges), i.e. E={(Tanya, Ahmed), (Bob, Mary),(Mary, Ahmed),(Bob, Ahmed)}.

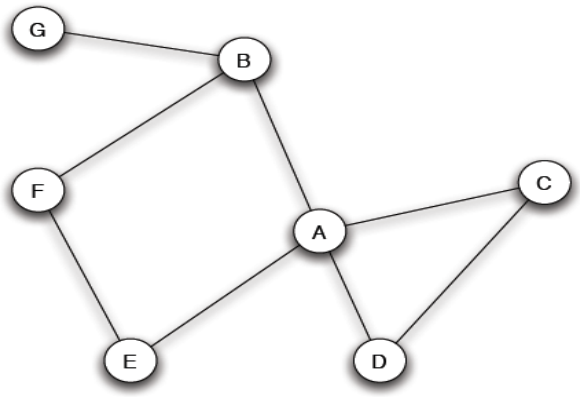


Figure 6 - An example of Social Network Graph
(Easley and Kleinberg, 2010. Page 48)

After modeling network data into graph, as shown in figure 6, social network mining tasks, such as (Kunegis et al. 2009), uses various graph based proximity measures, adapted from graph theory (Karamon et al. 2008). Following are some of the most commonly used graph-based properties used in mining and analysing network data. We use the example graph in figure 6 to explain these properties. The graph in figure 6 have 7 vertices as follows, V={A,B,C,D,E,F,G} and 8 edges as follows, E={(A,B), (A,D), (A,C), (A,E), (F,E), (F,B), (G,B)}.

a. Degree

The *degree* of a vertex in a graph can be denoted as *D (v)*, is the number of edges on that vertex. Let us consider the social network graph in figure 6. In this graph the degree of node A is 4 and node B is 3. The notion of *degree* in directed graph is a bit different. For example in the directed graph in figure 4 b, for the node B there are 2 types of degrees, namely *in-degree* which is 2, because edges from nodes A and D are directed towards node B. Similarly, the *out-degree* of node B is 1 as it has 1 edge, to node C, away from itself.

b. Bridge

An edge is called a bridge if deleting the edge would cause its nodes at each endpoints to lie in different connected components of a graph. For example the edge GB in figure 4 is a bridge because it will split the graph into two connected components.

c. Distance

The distance between two vertices in a graph (denoted as d_{xy}, the distance between node *x* and *y*) is the number of edges in a shortest path connecting them. For example the distance between nodes E and C in figure 6 is 2.

d. Closeness

Average distance from a node to all others. Closeness of node *x* can be calculated using following formula:

$$\frac{\sum_{y \in V} d_{xy}}{N-1}$$

Where N is the number of nodes in *G (V, E)*

e. Common Neighbours

For a node *x*, let $\Gamma(x)$ denote the set of neighbours of *x* in a social network graph. Common neighbours define *CN(x, y)* as the number of neighbours that *x* and *y* have in common:

$$CN(x, y) = \Gamma(x) \cap \Gamma(y)$$

For example number of common neighbours of nodes A and B in figure 6 is 1 (node D).

f. Clustering Coefficient

The clustering coefficient of a node is defined as the probability that any of its two selected neighbors connected with each other. That is, it is the fraction of pairs of a node's neighbors that are connected to each other by edges. Let K_v is

number of neighbours of vertex *v*. Then there is at most K_v * (K_v - 1)/2 number of edges that can exist between the neighbours of *v.* If L_v is the number of edges that actually exist between neighbours of *v* then clustering coefficient of node *v,* denoted as C(*v*) is:

$$C(v) = \frac{L_v}{K_v * (K_v - 1)/2}$$

For example, the clustering coefficient of node A in figure 6 is 1/6. Because number of neighbors of A is 4, so number of edges that can exist between it's neighbors is 4*(4-1)/2 = 6. And there is only one edge C-D among these possible six pairs.

Types of Social Network

The following are some of the main types of large-scale social networks that researchers have used for research in mining social network:

a. Friendship Network:

This is the simplest but most popular type of social network. Friendship network records who is friend to whom relationship among nodes. The largest of such network in online domain is currently Facebook which recently reported over 750 million users.

b. Collaboration Network:

Collaboration Network records who works with whom in a specific setting. Co-authorships among scientists, is an example of collaboration network. That is if 2 authors, A and B, publish a paper together there will be an edge between A and B in the corresponding Collaboration network.

c. Trust Network

Signed network or Trust network is a social network where we have both positive (e.g. *friendship*) and negative (e.g. *foes*) types of links or edges. E.g. in Wikipedia (www.wikipedia.com), one can vote for or against the nomination of others to adminship. Also in Epinions (www.epinions.com) users can express trust or distrust

of others (Leskovec et al. 2010a, 2010b). Figure 7 below is a graph model of a typical trust network.

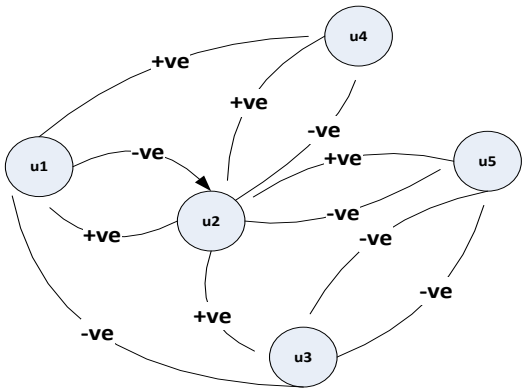


Figure 7 – A Trust Network graph example

d. Communication Network

Communication network models the "who-talks-to-whom" structure of social network. Such networks can be constructed from the logs of e-mail or from phone call records.

1.4 Social Network mining and Challenges

Data mining, such as classification, is very commonly used to tackle many SNA tasks and can be classified into two major categories; either *descriptive* or *predictive* (Figure 8).

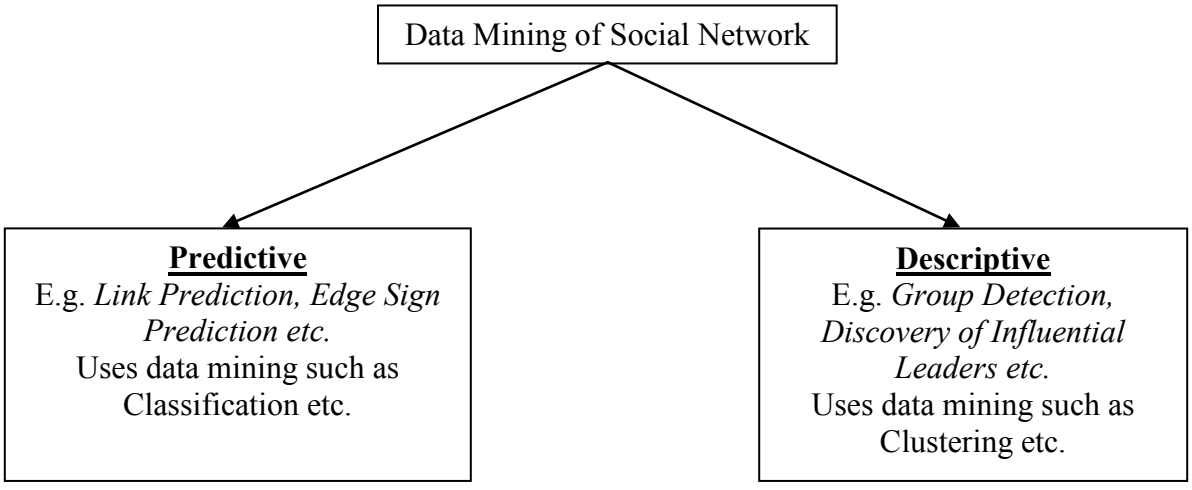


Figure 8 – Types of Social Network Mining Tasks

A *predictive* model produced using mining social network data makes a prediction about unknown values. For example Prediction whether an individual will be a friend of another individual. This task is also known as *Link Prediction* (Liben-Knowell and Kleinberg, 2003). Classification and Regression are commonly used to produce predictive model in Social Network. Some predictive mining of social network are listed below:

a. Link Prediction

Link prediction task in social network is to predict existence of an edge between two nodes. More formally given a snapshot of a social network at time *t*, link prediction task seeks to accurately predict which edges will be added to the network at time *t'* (Liben-Nowell and Kleinberg 2003). Liben-Nowell and Kleinberg (2003) used graph based properties such as common neighbours and distance as features in their dataset to apply classification mining techniques to predict future link. Tasker et al. (2003) in their work to predict link in social network of universities relied on using user's personal information such as music and book preferences etc. Link Prediction is an important component of Friend Recommendation system in many Social Network sites such as *Facebook.*

b. Node Classification

In large social network graphs, such as online social networks like Facebook, a subset of users or nodes may be labelled with information that indicate demographic values, interest, beliefs or other characteristics of the users (). Node Classification task make use of this data to predict or classify the labels of nodes of which the labels are unknown (Bhaghat et al., 2011).

A *descriptive* model identifies patterns or relationship, such as trends, clusters and anomalies etc. in data. An example of descriptive model of social network mining task can be identification or detection of groups within a Social Network. Clustering and pattern mining are some of the data mining techniques used for descriptive mining of social network. Some descriptive mining tasks in social network are listed below:

c. Community Detection (Clustering)

The goal of community detection task (Senator 2005) in social network mining is to detect groups or community in the social network graph which have more (or dense) edges among nodes in a same group than that of among nodes outside the group. There are many reasons to discover such group or communities in the network, for instance, target marketing schemes can be designed based on clusters, and it has been claimed that terrorist cells can be identified (Bonchi et al. 2011).

d. Influence Maximization

Using data mining method, influence maximization task attempts to help companies to determine potential customers to market to, so that by the virtue of mouth to mouth word spread, these customers can influence greater number of customers. Such marketing procedure is commonly known as *viral marketing*. For example Hotmail when launched grew from zero users to 12 million users in just 18 months on a very small advertising budget. There are many other application areas where *influence maximization* solution can be applied, such as virus or disease spread detection, social search, innovation adoption etc. Influence maximization in trust is the focus of this thesis.

Research challenges in Mining Social Network:

Privacy – Privacy protected mining of social network is a very important and sensitive issue that needs to be addressed (Wang et al. 2009) (Provost 2009). Current techniques to ensure privacy while mining social network is primarily based on *anonymization*, which is for example, replace node names with random IDs (Agrawal 2005).

Linked Data - While data representation and feature selection are significant issues for traditional data mining and machine learning algorithms, data representation for linked data is even more complex. Traditional data mining tasks such as, classification or association rule mining, usually attempt to extract knowledge from a dataset stored in a well-structured database (Getoor 2003) such as data in table 1 and 2. However in social

network, data are also linked to one another with using some type of relationship, such as *friendship.* So while analyzing a social network it is vital to take such relations under consideration. In many social network analysis (SNA) tasks, such as link prediction, graph based features- such as *degree*, is used and analyzed (Liben-Nowell and Kleinberg 2003). Identifying and computing such features and applying appropriate mining or machine learning technique is one of the challenges in SNA.

Scalability – In online domain the datasets are very large, with millions of nodes and edges. Mining tasks of these *large dataset* is quite time consuming especially if the task requires real time results. The challenge here is to devise efficient and scalable mining techniques that can process large amount of data in shortest possible amount of time and also produce models with high accuracy.

Dynamic One key drawback of recent studies, such as (Liben-Nowell and Kleinberg 2003), is that most of these work largely ignored the fact that social network is *dynamic*, meaning it changes over time (Tantipathananandh et al. 2007). There are significant work to be done in modeling and analyzing *dynamic* social networks.

1.5 Submodular Function Maximization

In recent times submodular function optimization has emerged as a fundamental problem structure in machine learning applications (Krause 2010). A submodular function is a set function defined as follows:

> **Definition 2** *Submodular Function* – A function $f: 2^X \rightarrow \mathbb{R}$ is submodular if for any $A \subseteq B \subseteq X$ and $x \in X \backslash B$, $f(B \cup \{x\}) - f(B) \leq f(A \cup \{x\}) - f(A)$.

The above definition basically states that any function *f* is submodular if its marginal gain (or benefit) for adding an element in a small set is larger than that of adding the same element to a larger set. This phenomenon is also known as *diminishing returns.* Sub modularity appears in many important problem settings including cuts in graph, set

covering problem and also in influence maximization problem (Feige et al. 2009). Optimization of any submodular function involves finding a set A which maximize or minimize $f(A)$. While minimizing a submodular function can be achieved in polynomial time, maximizing is unfortunately turns out to be very difficult (Feige et al. 2009). Indeed for all the problems mentioned above, the maximization is NP-Hard (Feige et al. 2009) (Kempe et al. 2003).

Submodular functions can be further classified as follows:

- Monotone functions: f is monotone if for any $A \subseteq B$, $f(A) \leq f(B)$
- Non monotone functions: no requirement as above.

The maximization of any submodular functions when it is monotone can be solved using a greedy algorithm which have approximation guarantee of (1-1/e) or 63% (Nemhauser et al. 1978). However when the function is non-monotone this guarantee does not hold. Feige et al. (2009) provides a deterministic *local search* based algorithm which have 1/3 approximation guarantee. Local search algorithms are commonly used to solve computationally hard optimization problems that can be formulated as finding a solution which maximizes a criterion, among a number of solutions. Local search algorithms typically start with a small solution based on certain criteria. Then, it applies local changes to the current solution, such as adding an element or removing an element, until a solution deemed optimal is found or certain criteria (example budget) is met. Feige et al., however, do not consider any restriction or *constraint* on the solution size. So the solution size can be potentially large. In many settings we require to maximize the function with constraint on the solution size. Such constraint is also known as cardinality constraint. For example in viral marketing we may want to target small group of people subject to a budget. That is we want set of influential set in which the number of influential nodes do not cross the budget. Lee et al. (2009) provides a different local search based algorithm for maximizing non monotone sub modular functions which can guarantees a solution with various kinds of constraint such as cardinality constraint. The local search based algorithm of Lee et al. (2009) proved to produce solution which is at least ¼ to the optimal solution under cardinality constraint.

1.6 Influence Maximization

Viral Marketing is the process of targeting the most influential users in the social network so that these customers can start a chain reaction of *influence* driven by word-of-mouth, so that with a small marketing budget a large population of a social network can be reached or *influenced*. Here 'influence' can be for a piece of information from government that needs to be spread to maximum possible members of social network, or it can be a new technology a company wants to promote etc. For example a phone manufacturer wants to promote their new phone model and have limited budget for the marketing campaign. To get maximum possible benefit out of the limited budget the company may want to choose a small group with largest influence, so that this small 'influential' group can influence greater number of potential customers. Selecting such influential nodes from a large social network graph is an interesting research challenge that has received a good deal of attention in the last years (Bonchi et al., 2011).

Influence according to Webster dictionary is "The power or capacity of a person or things in causing an effect in indirect or intangible ways". Several studies confirm that influence exists in online Social Network. For example, Leskovec et al. (2006, 2007b) show patterns of influence by studying person-to-person recommendation for purchasing books and videos, finding conditions under which such recommendations are successful. As considered in other literature on influence maximization such as Bonchi et al. (2010), in this thesis, *influence* is considered to be the ability (or probability) of a person to convince others to act or behave in certain way. This phenomenon or process of influence propagating from one user to another in a social network is also known as *influence propagation* or *diffusion process.*

The first to consider the propagation of influence and the problem of discovering influential users from a social network are Domingos and Richardson (2001, 2002). Kempe et al. (2003) formulated the same problem as a problem maximization of submodular function. They formalize *Influence Maximization* problem by adopting *diffusion models* (definition below), such as Linear threshold (LT) model and Independent Cascade (IC) model from mathematical sociology. We discuss these models in more details in chapter 2 of this thesis. Diffusion models, in general, models the spread

of influence or the diffusion process, through a social network represented by a directed graph *G*. In these models a node or user is said to be *active* if the node adopts a product (or performs an action) or *inactive* if the node does not adopt a product (or performs an action). We will discuss these models in detail in Chapter 2 of this thesis.

> **Definition 3** *Diffusion Model* - A diffusion model, also known as *propagation model*, describes the entire diffusion process and determines which nodes will be activated due the influence spread through the social network.

Based on these Kempe et al. (2003) defined *influence spread* function as stated below.

> **Definition 4** *Influence Spread* – Given a social network graph *G(V,E)*, a diffusion model *M* and an initial set of *active* vertices $A \subseteq V$, the influence spread of set A, denoted $\sigma_M(A)$, is the expected number of vertices to become active, under the influence of vertices in set *A*, once the diffusion process is over.

Note that both LT and IC models requires additional parameters, such as *influence probability,* along with social network graph *G(V,E),* to determine or compute influence spread.

> **Definition 5** *Influence probability* – Given a social network graph *G(V,E)*, influence probability, denoted as *p(u,v)* (such that $u, v \in V$ and $(u, v) \in E$) is the probability of node *v* to perform some action under the influence of user *u.*

Using these notations Kempe et al. (2003) defines the *k*-best influence maximization problem as follows:

> **Problem 1** - *Given a social network graph G=(V,E) along with influence probabilities of all edge in E, a diffusion model M and a number k such that* $k \leq |V|$*, find a set A such that* $A \subseteq V$, $|A| \leq k$ *and the influence spread, that is* $\sigma_M(A)$*, is maximum.*

From viral marketing perspective the parameter k is the budget of how many individuals we want to target and is given as input by the end user. And any algorithm that solves the above influence maximization problem must return a set of nodes consisting of k number of nodes. The selected set A is also referred to as "seed set".

Kempe et al. (2003) prove that the optimization problem (problem 1) is NP-hard for IC and LT diffusion models. This means that there is no polynomial time algorithm that can solve the influence maximization problem. However they show that $\sigma_M(.)$ function is sub-modular and monotone. Due to the sub modular property, when adding a node $v \in V$ of social network graph $G(V,E)$ into a seed set S, the incremental influence spread or *marginal gain* of influence spread function $\sigma_M(.)$, is smaller than adding v to any subset of S. Marginal gain of any node v in a social network with respect to any current seed set S, such that $S \subseteq V$ and v not in S, is the difference of influence spread caused by the node v. That is marginal gain for adding node v can be expressed as $\sigma_M(S \cup \{v\}) - \sigma_M(S)$. For example let us consider a seed set S and let's say it's influence spread is 7, i.e $\sigma(S) = 7$. Now let's also consider that adding a node v to S causes the influence spread to increase to 9, i.e $\sigma(S \cup \{v\}) = 9$. Then the marginal gain of node v is 9-7 = 2, i.e. $\sigma_M(S \cup \{v\}) - \sigma_M(S) = 9 - 7 = 2$. Therefore the sub modularity of influence spread function $\sigma_M(.)$ can be formally expressed as $\sigma_M(S \cup \{v\}) - \sigma_M(S) < \sigma_M(T \cup \{v\}) - \sigma_M(T)$ where $T \subseteq S \subseteq V$ and $v \in V$ of a social network graph $G(V,E)$. This actually means that under IC and LT models, the effect, or the marginal gain; of adding a new node to a seed set S is smaller than the effect of adding same node to a subset of S.

Note that submodular function f is said to be monotone if we have $f(S) \leq f(S \cup \{v\})$ for all elements v and for every sets $S \subseteq V$. That is adding an element to a set does not decrease the value of the function f.

According to Nemhauser et al. (1978) any submodular monotone function can be solved using natural greedy algorithm with a (1-1/e) approximation guarantee. That is due to the submodular and monotone property of $\sigma_M(.)$ function, the greedy solution will produce result which is at least 63% of the optimal. Thus, if A^* is an optimal set maximizing the function $\sigma_M(.)$ the approximation guarantee is expressed as:

$$\sigma_M(A) \geq (1 - 1/e) \cdot \sigma_M(A^*)$$

Kempe et al. (2003) present a greedy approximation algorithm applicable to IC and LT models. The algorithm requires computing marginal gain of influence spread of every node v, $\sigma_M(S \cup \{v\}) - \sigma_M(S)$, in each iteration. The node with highest marginal gain is 'greedily' added to the seed set until the size of the seed set reaches k. We will discuss the greedy algorithm further in Chapter 2 of this thesis.

The greedy algorithm for influence maximization requires 2 inputs (figure 9) as follows (Goyal et al. 2011):

a) A social directed network graph *G(V,E)* and

b) Influence Probability of each edge in *E*.

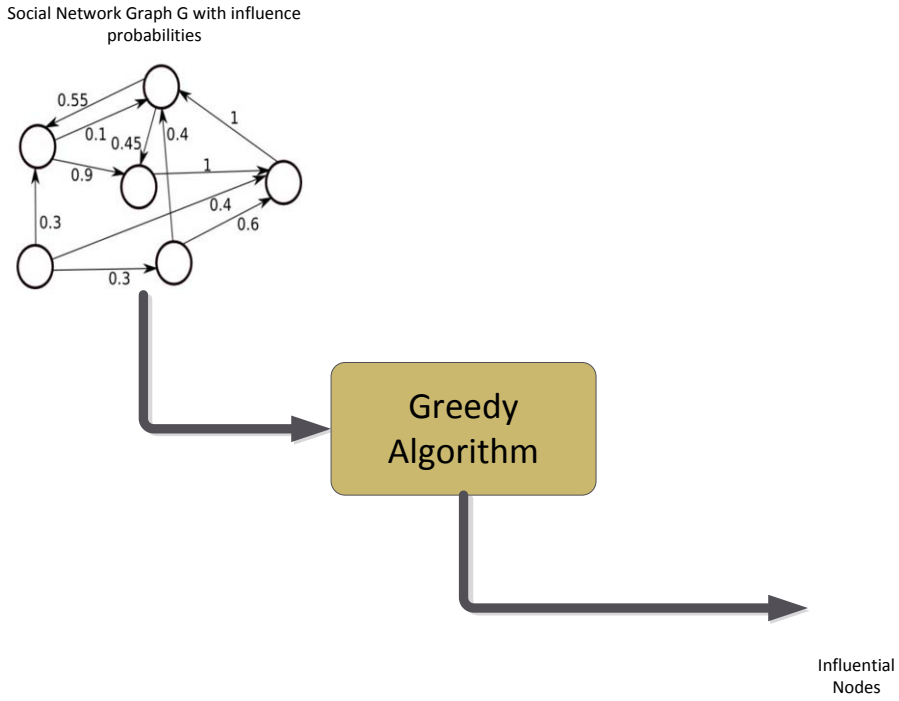


Figure 9: Greedy algorithm requires Social network graph with influence probabilities.

Although simple, the greedy algorithm of Kempe et al. (2003) is computationally expensive and not scalable to large social network. According to Chen et al. (2010), the greedy algorithm would fail or become unfeasible to extract influential nodes from large social network graph with more than 500K edges even on modern server machine. The computationally expensive step of the greedy algorithm is where we select the node that provides the largest influence spread. To be able to do this we need to compute influence

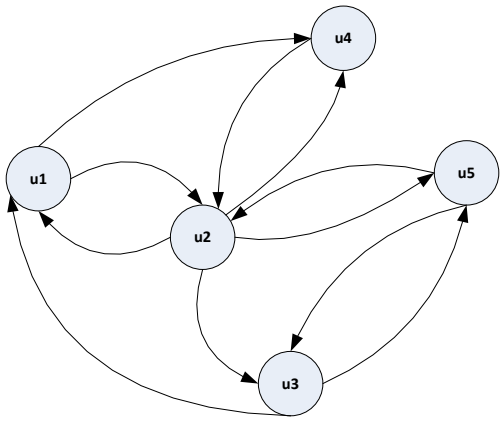
spread of each and every nodes which can be quite time consuming if the number of nodes and edges in the social network graph is very large. Most of the work, such as work of Leskovec et al. (2007) and Chen et al. (2009, 2010), that followed in the area of influence maximization attempt to tackle the efficiency issue of the greedy approach. We discuss some these works in Chapter 2 of the thesis. However, this is to be noted that tackling scalability is not within scope of this research and we keep the discussion on this issue limited.

Another major issue that is largely ignored in the works in influence maximization, such as (Kempe et al. 2003) and (Chen et al. 2010), is the question – *How and where we can compute Influence Probability?* Here, it is assumed that the network itself and influence probabilities are known and given to the algorithm as input. A social network graph G can be easily constructed if the data is explicitly available (Goyal et al 2011). However influence probabilities are not explicitly available. In most of the literature reviewed, influence probabilities are assumed to be given as input (Bonchi et al. 2011). To conduct experimentation researchers adopted various trivial methods of assigning influence probabilities (Bonchi et al. 2011). For example assuming *uniform* link probabilities (e.g. all link have probability p = 0.01), or the *trivalency* (*TV*) model where link probabilities are selected uniformly at random from the set {0.1, 0.01, 0.001}, or assuming the *weighted cascade* (*WC*) model, that is $p(u, v) = 1/d_v$ where d_v represent the in-degree of v (Goyal et al. 2011).

Goyal et al. (2011) recently compared these above methods and compared the different outcomes of the greedy algorithm. The finding of their experiments shows that the seed sets extracted under different probabilities settings are very different (with empty or very small intersection). This shows the importance of computing influence probability from real data instead of assigning them based on some naïve assumptions (Goyal et al. 2011). To tackle this issue researchers are now looking into ways to *mine* influence probabilities from Action Log of users in a social network (Goyal et al. 2010). Action log is a relation Actions(User, Action, Time) which contains tuples, for example (u, a, t), which indicates that user u (such that $u \in V$) performed action a, at time t (Figure 10b).

In (Goyal et al. 2010), factors such as the *influenceability,* of a specific user's tendency of getting influenced by its neighbours, or how influence drives a certain action are also

investigated. Finally, the authors also show that using the proposed method they can also predict *whether* a user will perform an action and *when* with high accuracy. We discuss this method further in Chapter 2 of this thesis.



User	Action	Time
u3	b	13
u2	b	13
u2	a	15
u1	b	15
u3	a	16
u5	b	16
u4	b	17
u5	a	18
u7	b	18
u4	a	19
u6	a	20

Figure 10: a) Social Network Graph. b) Action Log

Goyal et al. (2010) propose to apply their method as a preprocessing step of influence maximization algorithms, such as Greedy by Kempe et al. (2003) or CELF by Leskovec et al. (2007). The overall framework of this method can be demonstrated by figure 11. In this framework Goyal et al. (2010) suggests to first learn influence probabilities from social graph and propagation log (or Action Log) and then apply Greedy (Kempe et al. 2003) or Lazy optimization (Leskovec et al. 2007b) algorithm to discover the seed set. We discuss these methods further in Chapter 2 of the thesis report.

Note that all of these works consider only *positive* influence among users in a social network. That is techniques for influence maximization (Kempe et al. 2003) and mining influence probability (Goyal et al. 2010) only consider how much a node has influence on another node to perform a certain task. However in real life scenario a node can also have some degree of *negative* influence on another node, especially by a user who he/she does not trust.

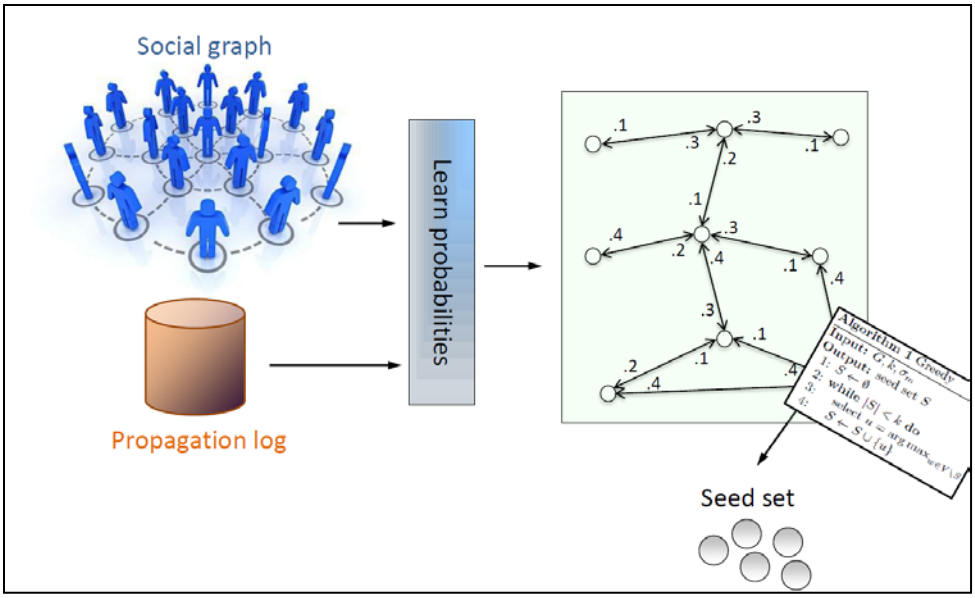


Figure 11: Proposed framework by Goyal et al. (2010)

(Bonchi et al. 2011. Pg. 11)

Therefore in this research our goal, though same as the works discussed above (i.e. discovering influential nodes), but different in the sense that we consider both *positive* and *negative* influence among users in a *trust network* to discover influential nodes. In the following section we summarize the problem definition and outline the contributions of the proposed research.

1.7 Thesis Contribution

Recent research shows that people trust the information they obtain from their close friends or family more than information obtained from general advertisements (Chen et al. 2010). Furthermore viral marketing is different from other strategies of marketing, because it is based on *trust* among such close social circle of families and friends (Chen et al. 2010). Therefore we claim that for influence maximization we need to consider negative influence by a distrusted user while modeling diffusion process. That is, we should consider a user can also get *negatively* influenced by another user, especially by a *distrusted* user. Existing diffusion models such as Linear Threshold Model (Kempe et al. 2003) and influence maximization techniques do not consider negative influence and resulting seed sets may contain negative influencers. Based on this in this research we

consider trust social network where relationship is either *trust* or *distrust.* Such relationships in trust networks are *asymmetric,* meaning an existence of an edge (*u,v*) does not necessarily mean existence of an edge (*v,u*) (Guha et al. 2004). For example in a trust network, as defined in section 1.3, a positive edge (*u,v*) signifies that node *u* trust node *v,* but *not* vice versa. There are several examples of trust network in online domain For example, users on Wikipedia (www.wikipedia.com) can vote for or against the nomination of others to adminship; users on Epinions (www.epinions.com) can express trust or distrust of others; and users on Slashdot (www.slashdot.com) can declare others to be either "friends" or "foes" (Leskovec et al. 2010). Existing diffusion models for IM are modeled such a way that a node's probability of performing an action (or adopting a product) will increase as the number of his/her friends performing the same action increases. However, we argue that, a node's probability of performing an action (e.g. Buy iPhobe 4S) should also *decrease* if its distrusted users, also buy iPhone 4S.
Motivated by this the formal problem definition we propose to tackle is as follows:

Thesis Problem Definition – Find Influential Nodes from a directed trust network graph, *G*(*V,E*) with every edge (*u,v*) of E is directed and labelled either positive (trust) or negative (distrust), and Action Log, Actions(User, Action, Time) such that every user *u* in User column of action log table is member of *V*.
To solve the above problem we make the following contribution in this research:

1) We propose a new diffusion model named Trust-General Threshold (TGT) model which incorporates both positive and negative influence probabilities based on trust relationship among users in trust network.

2) Based on this new TGT model we propose a new influence maximization framework for trust network, called Trust-Influential Node Miner (T-IM), which takes trust network data and action log to find influential nodes.

2) To compute influence probabilities we propose an algorithm (Action Pattern Generator) which mines the action log to find frequent patterns of action performed by

trusted and distrusted users and use it to compute both positive and negative influence probability using Bernoulli distribution.

3) We show that approximation guarantee of (1-1/*e*) by existing IM algorithms such as Lazy Forward by Leskovec et al. (2007) is not applicable to TGT model because the influence spread function in this model is non monotonous.

4) We also show that influence spread function is still sub modular and define the problem of finding influential nodes from trust network under TGT model as an *optimization of non-monotone submodular function problem*.

4) We propose a new algorithm, mineSeedLS, based on local search (Lee et al. 2009) to solve IM under our proposed TGT model.

5) Perform in depth analysis of our proposed solution using real life data set collected from Epinions (www.epinions.com) and Wikipedia (www.wikipedia.com). In terms of number of quality of seed selected, our experiments shows that mineSeedLS outperforms greedy based solutions by almost 35%.

CHAPTER 2

RELATED WORKS

Analyzing *information diffusion* and *social influence* in social network has many applications to real-world. Influence maximization for viral marketing is an example of such an important application (Tang et al. 2009). In this section, we introduce the problem of influence maximization and review recent research progress. In section 2.1 we introduce some vocabulary related to information diffusion (Gruhl et al. 2004) process and also discuss several diffusion models that attempts to describe the diffusion process in social network. Most works, such as (Kempe et al. 2003, Leskovec et al. 2007b, Chen et al. 2010) on influence maximization problem rely on these models. In section 2.2 we discuss the classical paper by Kempe et al. (2003), where they first formulated the influence maximization as a discrete optimization problem and solved it using a greedy algorithm. In section 2.3 we discuss 'Lazy forward' optimization, by Leskovec et al. (2007), which is about 700 times faster than greedy of Kempe et al. (2003). In section 2.4 we discuss further improvement to the greedy approach mainly in terms of scalability. In section 2.5 we discuss some of the more recent data mining based approaches, such as by Goyal et al. (2010) in this area.

2.1 Diffusion Models

In information diffusion process in social network there exists some vocabulary specific to the context. In a social graph G(V,E), a vertex $v \in V$ becomes (or is called) *active*, if the information has reached the vertex and was accepted by it. Similarly a vertex which information has not reached or got convinced so far, is called *inactive*. It is assumed that during the process of diffusion of the information, an inactive vertex can become active but not vice-versa. To simplify diffusion models this restriction is applied. In order to diffusion process to start, there must be some initial set of active nodes called *seed nodes* which are initially activated.

A diffusion model, also known as *propagation model*, describes the whole diffusion process and determines how the influence propagates through the network. The role of these diffusion model is primarily, to replicate or simulate real life diffusion process and determines which nodes and how many nodes will be activated by any given set of nodes (called *seed nodes*) after the diffusion process is over. There are few classical models which are used very commonly used to tackle the influence maximization problem. Here, we review some of them.

The status of the chosen set of users to market (also referred to as "seed nodes") is viewed as *active* and initially all other users are considered *inactive*. Then, the chosen activated users, may further influence their friends (neighbour nodes) to be active as well. In this section we discuss the two of the most well known models namely *Linear Threshold Model* (LT) and *Independent Cascade Model* (IC) (Kempe et al. 2003). These models, or their variations, are most commonly used diffusion models in Influence Maximization.

2.1.1 Linear Threshold Model

Given a directed graph of social network *G(V,E).* A threshold value θ_v is assigned to each node *v* in *V.* Also each edge (*u,v*) in *E* is also assigned with a weight value $b_{u,v}$. In linear threshold model the probability of a vertex *v* to become active will increase as more of its neighbours become active. The vertex *v* is influenced by each of its neighbour *w* according to the weight $b_{w,v}$ such that sum of all the weights $b_{w,v}$ for all $w \in N_v$ is ≤ 1. Where N_v is set of all active neighbours of *v*. At any given time *t* a vertex become active if

$$\sum_{w \in N_a(v)} b_{w,v} \geq \theta_v$$

Where $N_a(v)$ is the set off all active neighbours of *v* at time t-1. Usually threshold θ_v is uniformly set at random in the interval [0,1]. However in some approach this threshold is simply fixed to a given value for all the vertices, such as 0.7. (Kempe et al. 2003)

The LT Model can be summarized using the following steps:

Given

- Threshold θ_v
- Seed Sets A $\subseteq$ V

In time *t*

- Vertices that are active at time t-1 will remain active
- Inactive vertex *v* becomes active at time t+1 if

$$\sum_{w \in N_a(v)} b_{w,v} \geq \theta_v$$

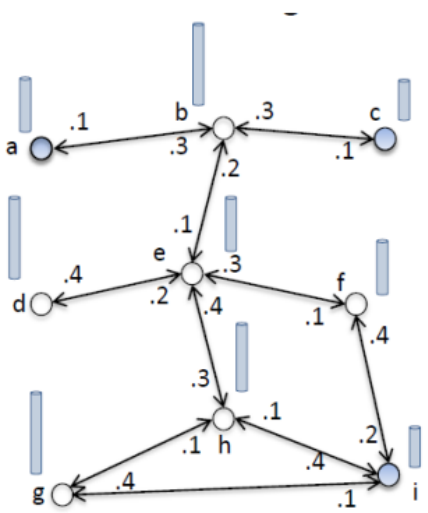
- STOP when no nodes becomes active

Example:

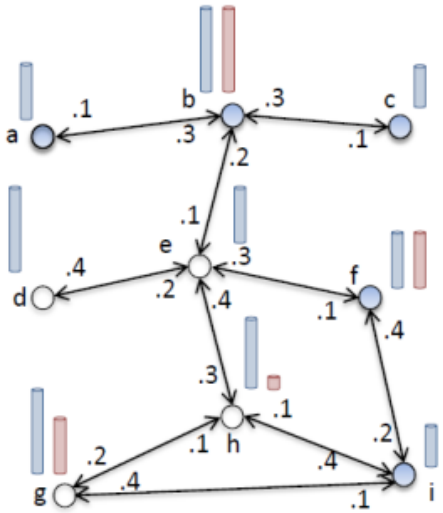
Let us now look at an example of an LT model. Let us consider the social network graph at time *t*-1 as shown in figure 12.a below. Let us also consider $A = \{a, c, i\}$, meaning nodes *a, c* and *i* are initially activated. The left column bar on each node represents the threshold θ and each directed edge has a weight value $b_{w,v}$. For example $b_{i,h}$ is .1 and $b_{a,b}$ is .3.

At time *t* (Figure 12.b) node *b* be will be active as $b_{a,b} + b_{c,b} = 0.3 + 0.3 = 0.6 \geq \theta_b$. Similarly node *f* becomes active at time *t* as $b_{i,f} = 0.4 \geq \theta_f$. Note here that nodes *g* and *h* do not get activated here as the total weight of active neighbours (right column bar) is less than their respective threshold θ.

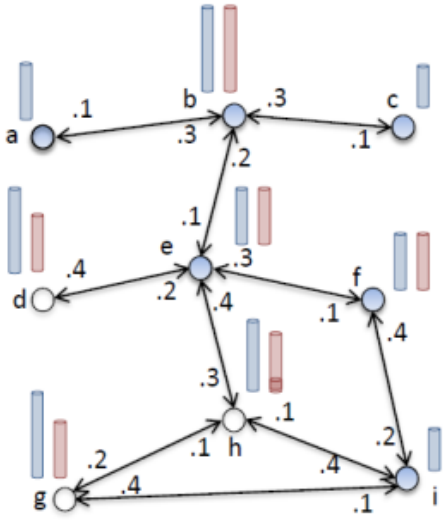
At time *t*+1 (Figure 12.c) node *e* gets activated as total weight of active members *b* and *f* is $b_{b,e} + b_{f,e} = 0.1 + 0.3 = 0.4 \geq \theta_e$. The diffusion process stops at this point as there are no more nodes that can be activated



a) At Time *t-1*



b) At time *t*



c) At time *t+1*

Figure 12 – Linear Threshold Model example.

2.1.2 Independent Cascade Model

In independent cascade model every arc (u,v) in $G(V,E)$ is associated with the *influence probability* $p(u,v)$ of u influencing v. Influence Probabilities is the probability of a node to be influenced by another node. At time t, nodes that became active at t-1 will activate their inactive neighbours according to probability $p(u,v)$. The IC Model can be summarized using the following steps:

```
Given
   -  Seed Sets A ⊆ V
At time t
        If u becomes active at time t-1
           u attempts to activate each of its inactive
neighbours v.
           u activates v with probability p(u,v)
           If successful, v becomes active in step t+1
STOP when no nodes becomes active
```

Example:

Let us now look at an example of an IC model. Given a social network graph at time *t-1* as shown in Figure 13.a below. Let us consider $A=\{a,i\}$, meaning nodes a and i are initially activated. Each directed edge $(u,v) \in E$ has a influence probability value $p(u,v)$. For example $p(a, b) = .3$ means that the probability of node u influencing node v is 30%. At time t-1 the node i will attempt to activate g, h and f and will succeed with the probability of .4, .1 and .4 respectively. Let say it succeeds to influence g and f and fails to influence h. Similarly node a attempts to influence node b and succeeds. So node g, f and b are activated at time t (Figure 13.b). And this process continues until no more nodes can be activated (Figure 13.c and 13.d).

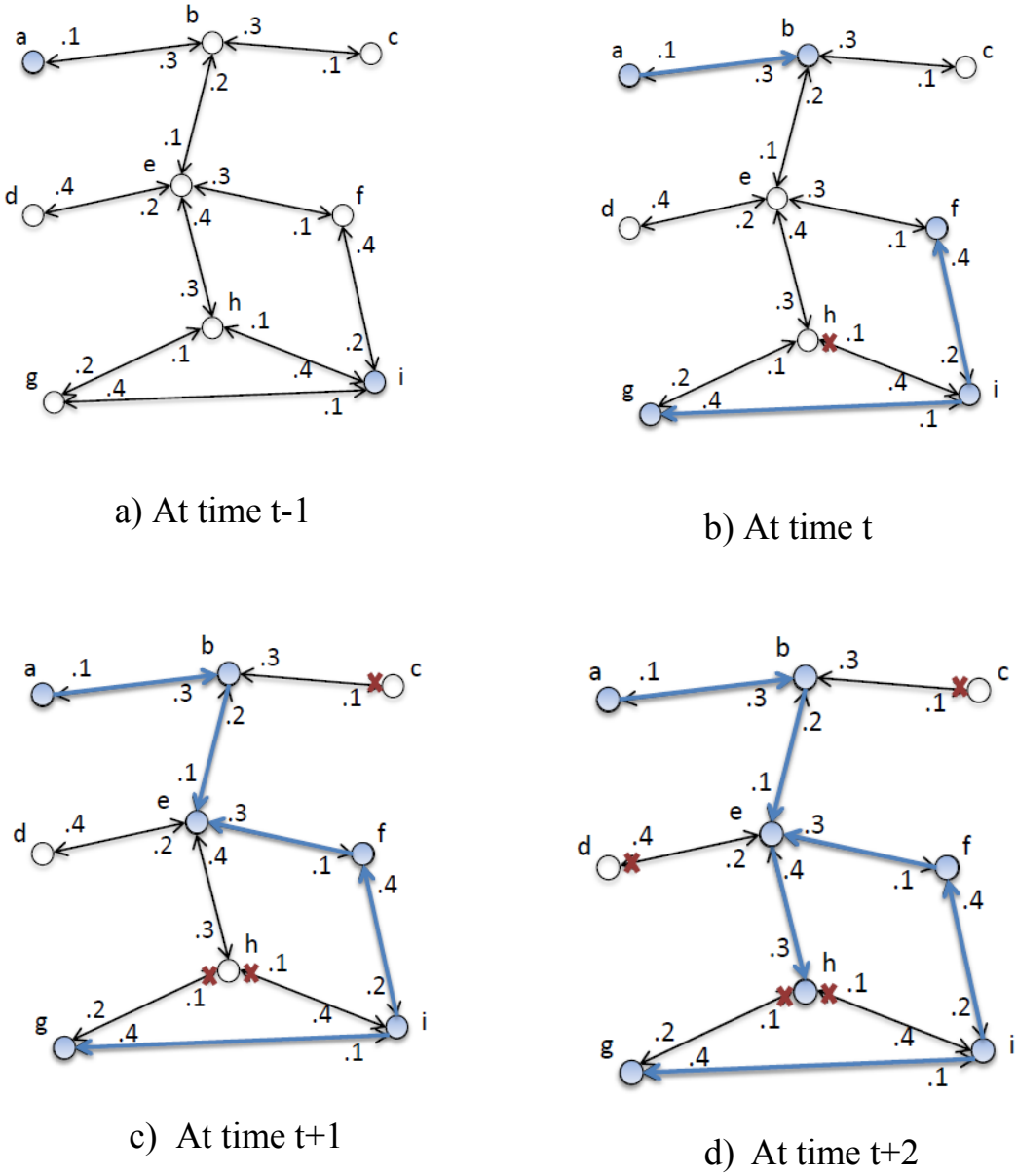


Figure 13 – Independent Cascade Model example.

2.2 Greedy algorithm for Influence Maximization

Kempe et al. (2003), studied the Viral Marketing problem in several of the most widely used models in Social Network Analysis such as *Linear Threshold Model* and *Independent Cascade Model as* discussed is Section 2.1. Their goal is to formally express

the viral marketing problem, choosing a good initial set of nodes (customers) to target, as optimization problem in the context of these models.

First they introduced diffusion models namely LT and IC (Section 2.1). Then they defined influence spread function, $\sigma(.)$ as follows, given a network graph $G(V,E)$ which is directed with influence probability or weight for each edge as shown in figure 14, and a diffusion model M, the influence of set of vertices $A \subseteq V$, denoted $\sigma_M(A)$ is the expected number of active vertices once the diffusion process is over. Using these notations we can formally define the k-best maximization problem as follows:

> **Problem 1 (Influence Maximization)** Given a social network graph $G(V,E)$ along with influence probabilities of all edge in E, a diffusion model M and a number k find a set $A \subseteq V$, $|A| \leq k$ such that influence spread, that is $\sigma_M(A)$, is maximum.

Kempe *et al.* (2003) prove that the optimization problem is NP-hard for both LT and IC Models. That is influence maximization problem as defined above cannot be solved in polynomial time. However they showed that $\sigma_M(.)$ function is sub-modular and monotone.

> **Theorem 1(Kempe et al. 2003):** For an arbitrary instance of the Independent Cascade Model or Linear Threshold Model, the resulting influence function is submodular and monotone.

According to Nemhauser *et al.* (1978) any submodular monotone function can be solved using natural greedy algorithm with a (1-1/e) approximation guarantee (Theorem 1).

> **Theorem 2(Nemhauser et al. 1979):** The greedy algorithm gives a (1 – 1/e) approximation for the problem $max\{f(S) : |S| < k\}$ where f is a monotone submodular function.

That is due the submodular and monotone property of $\sigma_M(.)$ function, the greedy solution will produce result which is at least 63% of the optimal. They presented the greedy algorithm (figure 15) which takes social network graph $G(V,E)$, k and Model m. It begins

by initializing seed set S to Null (line 1). Vertex w which maximize the marginal gain $\sigma_M(S \cup \{w\}) - \sigma_M(S)$ is added to S at each iteration (line 3), until the $|S|=k$.

Table 5: Sample social network data with Influence Probability

From Node	To Node	Influence Probability
A	E	0.1
A	F	0.9
B	A	0.3
B	D	0.4
B	C	0.3
C	D	0.6
C	E	0.4
D	E	1
E	A	0.55
E	F	0.45

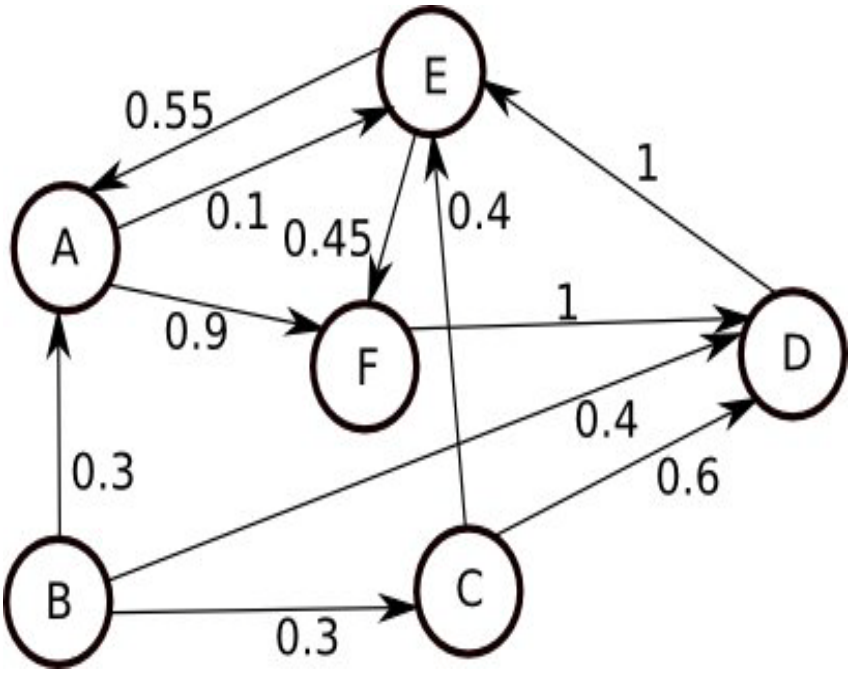


Figure 14 - Social Network Graph with influence probability modeled on data in table 5.

Algorithm 1: The Greedy k-best influence maximization algorithm

Input: *G, k,* σ_m */*G is the social network graph, k the desired size of the seed set and* σ_m *is the influence model*/*

Output: Seed set *S*

Begin

1. Set $S \leftarrow \emptyset$
2. **for** $i = 1$ to k **do /***Look for seeds until *k* seeds are found /
3. $u \leftarrow \text{argmax}_{w \in V-S}(\sigma_m(S \cup \{w\}) - \sigma_m(S))$*;/Pick node u which have maximum marginal gain/*
4. $S \leftarrow S + u$
5. **end for**

Figure 15: The Greedy k-best influence maximization algorithm

For Step 3 of the greedy algorithm, the conventional method for estimating all the marginal influence gain of any node w in V, $\sigma_m(S \cup \{w\}) - \sigma_m(S)$, with respect to current seed set S is described as follows (Kempe et al. 2003). First, a sufficiently large positive integer M is specified. For any node $w \in V - S$, the process of the diffusion model (IC or LT model) is run for the initial active set S and also for $S \cup \{w\}$, and the number of final active nodes activated by S (or $S \cup \{w\}$), denoted as $\phi(S)$ (or $\phi(S \cup \{w\})$) , is counted. Each $\sigma(S)$ and $\sigma(S \cup \{w\})$ is then estimated as the empirical mean obtained from M such simulations. This process of estimation is known as Monte Carlo simulations (Kimura et al. 2007). The conventional method for estimating $\sigma(S \cup \{w\})$ for all $w \in V - A$ as follows (Kimura et al. 2007):

INPUT: A Seed Set S

OUTPUT: Average number of nodes activate by S ($\sigma(S)$)

1. For m = 1 to M do
2. Compute $\phi(S)$ /*Compute # of users activated by S/*
3. Set $x_m \leftarrow \phi(S)$
4. End For
5. Set $\sigma(S) \leftarrow (1/M)\sum_{m=1}^{M} x_m$ /*Return the average # of users activated by S/*

Here, each $\phi(S)$ is computed as follows (Kimura et al. 2007):

1. Set $H_0 \leftarrow S \cup \{w\}$. /* H_0 is set of currently active users*/
2. Set $t \leftarrow 0$. /*Set time t to 0*/
3. while H_t is not $\emptyset$ do /*Until no new nodes are activated*/
4. Set $H_{t+1} \leftarrow$ {the activated nodes at time t + 1}.
5. Set $t \leftarrow t + 1$.
6. end while
7. Set $\phi(S) \leftarrow \sum_{j=0}^{t-1} |H_j|$ /Return total number of activated nodes by S/

Example:

Now let us consider the social network graph with influence probability given in figure 14 above. Using this we will demonstrate the greedy algorithm under independent cascade model. For simplicity we will demonstrate the algorithm by setting M to 1. Meaning we are going to estimate the influence spread by running the diffusion random process only once. Also let us consider *k* to be 2 i.e. we are looking for a set of influential nodes, S of size 2 from the social graph in figure 14. First the algorithm will first initialize S to $\emptyset$ (Line 1). Then the algorithm will enter a *for* loop. Since k=2 this loop will run twice. In the first iteration the algorithm will look for node $w \in V \backslash S$ which maximize the marginal gain of influence spread relative to the set S (is null at this point).

To get this algorithm will compute the number of nodes activated by set $S \cup \{w\}$ for each $w \in V \setminus S$ under the independent cascade model. Following is the list of all nodes $w \in V \setminus S$ and number of nodes that gets activated by each of these:

A – 4 as it activates nodes F, D and E
B – 3 as it activates nodes D and E
C – 3 as it activates nodes D and E
D – 2 as it activates node E
E – 1 as it does not activate any more nodes
F – 3 as it activates nodes D and E

Based on the above, the algorithm will choose node A in the first iteration and put it into set S. Now the seed set S = {A} and we still need to find one more node so that |S| = 2.
In the second iteration the algorithm will compute marginal gain, $\sigma_M(S \cup \{w\}) - \sigma_M(S)$ for each $w \in V \setminus S$, of each of the remaining nodes in V\{A}. Following is the list of nodes and its corresponding number of nodes activated by adding it to set S:

B – 2 as it activates node C
C – 1 as it does not activate any additional nodes.
D – 0 as it does not activate any additional nodes and D is already activated by set S
E - 0 as it does not activate any additional nodes and E is already activated by set S
F - 0 as it does not activate any additional nodes and F is already activated by set S

From the above numbers we can see that B has the highest marginal gain, i.e. activates most nodes. So node B is now added to set S which now has 2 nodes {A,B}. Since |S|=2 which was our required number of influential nodes the algorithm stops and here and return S={A,B}.
To test their approach Kempe et al. (2003) used co-authorship data compiled from the complete list of papers in the high-energy physics theory section of the e-print arXiv (www.arXiv.org). They modeled the data as a collaboration graph which contains a node for each researcher who has at least one paper with co-author(s) in the arXiv database.

The resulting graph has 10748 nodes and 53000 edges. They compared their algorithm in three different models of influence – independent cascade model, the weight cascade model, and the linear threshold model. Also they further compared their greedy algorithm with heuristics based on node's degrees and centrality within the network, as well as choosing random nodes to target. The authors claim to have shown through experiments that their greedy algorithm significantly outperforms, in terms of influence spread, the degree and centrality-based heuristics in influence spread.

One of the main limitations of the above greedy approach is efficiency and scalability. Note that for selecting a node (step 3) that maximize the marginal gain $\sigma(S \cup \{w\}) - \sigma(S)$ is computationally expensive, as it needs to explore all the possible combinations. Kempe et al. (2003) used Monte Carlo simulations, as discussed above, of the propagation model for sufficiently many times to obtain an accurate estimate (Goyal et al. 2011). As a result, finding a very small seed set in a relatively large network (e.g. 15000 vertices) could take days to complete in a modern server machine (Chen et al. 2009). Several recent studies aimed at addressing this *efficiency* and *scalability* issues such as by Leskovec et al (2007) and Chen et al. (2009, 2010).

2.3 'Lazy Forward' Optimization

The most notable work in attempt to improve the scalability of greedy approach of influence maximization is by Leskovec et al. (2007). Leskovec et al. in (Leskovec et al. 2007b) tackle the problem of outbreak detection, which is the problem of selection of nodes in a network in order to detect the spreading of virus or information as quickly as possible. Though this work is not exactly in the area of influence maximization, the work done here contributed towards improving the scalability.

The authors refer to the work of Kempe et al. [2003], and suggest that this work generalizes the work on selecting nodes which maximize the influence in social network**.** Leskovec et al. (2007) exploited the submodular property of influence function $\sigma_m(.)$ to develop an efficient algorithm called CELF, based on a "lazy-forward" optimization in selecting seeds. Due to the submodular property the marginal gain of a node in the current iteration of the greedy algorithm cannot be better than its marginal gain in the previous iterations.

To take advantage of this property CELF algorithm maintains a table of marginal gain, mg(u,S), of each node u in current iteration sorted on mg(u,S) in decreasing order, where S is the current seed set and mg(u,S) is the marginal gain of u with respect to S. Table mg(u,S) is re-evaluated only for the top node in next iteration. If required the table is resorted. If a node remains at the top after this, it is picked and added to the seed set.

Leskovec et al. (2007) evaluated their methodology extensively on two large scale real world scenarios namely: a) detection of contamination in large water distribution network, and b) selection of informative blogs in a network of more than 10 million posts. Using these scenarios they compared CELF's performance and scalability with natural greedy algorithm as shown in figure 16. In terms of performance, CELF generated results that are at most 5% - 15% from optimal. In terms of scalability, CELF also performed a lot better than greedy. For example for selecting 100 influential blogs, the greedy algorithm require 4.5h, while CELF takes 23 second (about 700 times faster). Also memory usage of CELF is about 50 MB compared to 3.5 GB required for greedy algorithm.

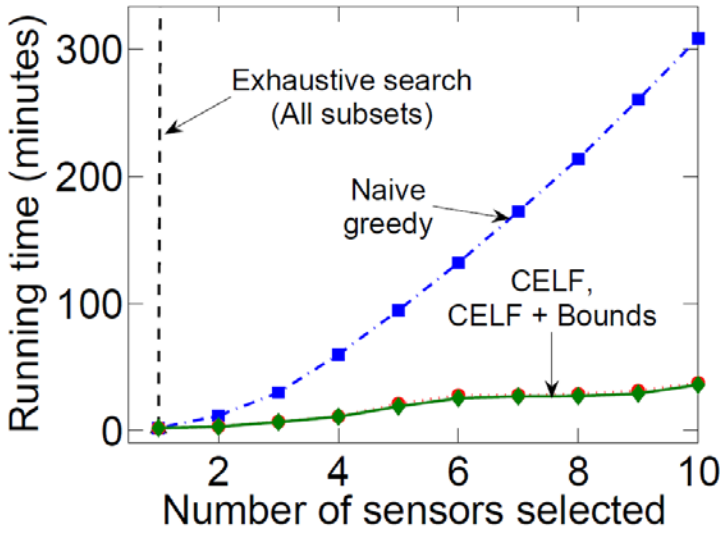


Figure 16. Running time of exhaustive search, greedy and CELF. (Leskovec et. al. 428)

Example:

Let us consider the same example we discussed in the previous section. Consider the social network graph in figure 14 with given influence probabilities. Again let us set k=2, ie we are looking for the seed set of size 2. Similar to the greedy approach CELF optimization will pick node A in the first iteration and will also create a table mg(u,S) as follows:

Mg(A,{})	4
Mg(B,{})	3
Mg(C,{})	3
Mg(D,{})	2
Mg(E,{})	1

As mentioned the algorithm will pick A as its marginal gain is the highest and will be removed from the table as follows:

Mg(B,{})	3
Mg(C,{})	3
Mg(D,{})	2
Mg(E,{})	1

Now in the next iteration the CELF optimization the algorithm will only evaluate the top node, ie node B. We saw earlier that marginal gain of node B in respect to S={A} was 3. As there is no change the node B will selected as next seed and added to the seed set S. Note that unlike greedy algorithm discussed in the previous section the CELF algorithm avoids computing marginal gain of rest of the nodes (such as C, D and E) and still gets the same result. Thus CELF is much efficient compared to greedy algorithm. Leskovec et al. (2007) empirically show that CELF improves the efficiency of the greedy algorithm by about 700 times.

2.4 Improving scalability of Greedy

2.4.1 MixedGreedy

Chen and Yang (2009) attempts to reduce run time of the greedy algorithm of Kempe et al. (2003) and its improvement by Leskovec et al. (2007). The authors point out that the work of Kempe et al. (2003) is not at all efficient. They claim that it would take days to find a small seed set in a moderately large network (e.g. 1500 vertices). Also though they acknowledge that the CELF algorithm by Leskovec et al. (2007) is 700 times faster, but they claim that it still takes few hours to complete in graph with few ten thousand nodes. Chen et al. (2009) tackle the efficiency issue of influence maximization by introducing new schemes to improve the greedy algorithm and combine it with CELF (named MixedGreedy) to obtain more efficient algorithm.

They conducted extensive experiments on two real life collaboration networks to compare their proposed approaches with CELF optimization. Experimental results showed that the for their new greedy algorithm the influence spread was exactly that of natural greedy algorithm of Kempe et al. [2003] and running time was 15-34% less than CELF.

2.4.2 CELF++

Goyal et al. (2011a) propose CELF++, based on CELF by (Leskovec et al. 2007b), which exploits the sub modularity of ininfluence spread function. It avoids unnecessary re computations of marginal gains incurred by CELF. Goyal et al. conducted experiments to test CELF++, in IC model, using two real world dataset from collaboration networks collected from arXiv (www.arXiv.com). The authors claim that CELF++ was about 35-55% in all the datasets. Furthermore the memory usage of CELF++ was more than that of CELF but not significant. The authors further claim that CELF++ works efficiently and effectively, and is significant improvement in terms of both running time and average number of node look up.

2.4.3 SimPath

Goyal et al. (2011b) attempt to design a scalable algorithm delivering high quality seeds for influence maximization problem under LT model. In Goyal et al. (2011) the authors highlighted several performance related drawbacks of simple greedy approach of Kempe et al. (2003) which are as follows: i) it requires to run Monte Carlo (MC) simulations sufficiently many times to estimate accurate influence spread which prove to be very expensive. ii) the greedy algorithm makes O(nk) calls to the MC simulations where n is the number of nodes in the graph and k is the number of seeds to be picked. They acknowledge the CELF optimization of Leskovec et al. improves the greedy, but stated that it still found to be quite slow and definitely not scalable.

To address these issues Goyal et al. propose the SIMPATH algorithm for influence maximization under the LT model. SIMPATH is built using CELF optimization that iteratively selects seeds in a lazy forward manner. SIMPATH make use of two key ways of optimizing the computation and improving the quality of seed selection which are as follows: i) Vertex Cover Optimization ii) Look Ahead Optimization.

For experiment they used four real-world datasets to evaluate SIMPATH and compare it with other well known influence maximization algorithms. The goal here was to evaluate in terms of efficiency, memory consumption and quality of the seed set.

The authors claim that qualities of seed sets, in terms of influence spread, generated by SIMPATH were quite competitive. For instance SIMPATH was only 0.7% lower than CELF in spread achieved and performed better compared to all other algorithms. In terms of efficiency, SIMPATH was fastest among all the algorithms; expect HIGH-DEGREE and PAGERANK of Page et al. (1998). Based on these results the authors claim that SIMPATH outperforms LDAG of Chen et al. (2010), in terms of running time, memory consumption and quality of seed sets.

2.4.4 Community-Based Greedy

In (Wang et al. 2010) the authors attempt to reduce the computational cost of greedy algorithm of influence maximization problem by using Community-based approach. Their focus is on mobile social network, where individual communicate each other using mobile phones. The authors highlight that most of the Greedy based approach is

computationally expensive and not suitable for a large mobile network. Also they mention that none of the previous works attempted to use community based approach for influence maximization. Wang et al. (2010) in their work proposes a new algorithm for mining top-K influential nodes, called Community-based Greedy algorithm (CGA). The idea behind their approach is to exploit the community structure property of social networks. CGA consists of two components 1) an algorithm for detecting communities based on information diffusion and 2) a dynamic algorithm to find influential nodes from these communities. To evaluate the effectiveness and efficiency of the proposed CGA algorithm the authors used data sets collected from call detail record (CDR) from China Mobile. They compared run time and influence spread of CGA with MixedGreedy of Chen et al. (2009), NewGreedy of Chen et al. (2009) and DegreeDiscount of Chen et al. (2010). The authors claim that the run time of CGA was faster than MixedGreedy but as expected was slower than heuristics based algorithms. However the experimental results showed that the influence spread of CGA was very close to MixedGreedy and NewGreedy. CGA outperformed the rest of the heuristic based algorithms quite comfortably. The authors claim that their approach is more than an order magnitude faster than the state of the art Greedy algorithm for finding top-K influential nodes.

2.4.5 Sparsification of Influence Network

In (Mathioudakis et al. 2011) the authors tackle the scalability issue of influence maximization problem by pruning the social network graph, called sparsification of network, while preserving the most of its properties. This work can be collocated with works on network simplification, the goal of which is to identify sub networks that preserve properties of a given network. They argue that such simplification of social network will yield significant improvement in terms of scalability. The authors define the problem of sparsification in terms of observed activity in the network. Given a social network and log of actions performed by nodes in the

network, the problem is to find a sub network of prefixed extent while maximizing the likelihood of generating the propagation traces in the log. They prove the problem to be NP-Hard. They define a greedy algorithm to solve this, called SPINE (Sparsification of influence network). SPINE works in two phases as follows: First it selects a set of arcs

that yields a finite log-likelihood. Then it greedily seeks a solution of maximum log-likelihood.

They used two real world data sets collected from Yahoo! Meme and a prominent online news site. To test their hypothesis that SPINE could play important role in reducing run time of influence maximization problem, they applied SPINE as preprocessing step to see if computation on sparsified network gives up any accuracy. For each network generated for collected datasets they measured the running time and influence spread before and after sparsification. Mathioudakis et al. (2011) claim that the experimental results shows that run time on sparsified network is considerably low compared to the full network. Also influence spread of seed set computed using sparsed network is quite close to that of the full network.

2.5 Data Mining Approaches

As highlighted earlier both greedy of Kempe et al. (2003) and its improvement, CELF by Leskovec et al. (2007) requires two kinds of data – a directed graph G, and assignment of probabilities or weights (depending on which diffusion model is considered) to the edges of G. A social network graph G can be easily constructed if the data is explicitly available. However influence probabilities or weights as used in LT and IC models are not explicitly available. In the work of Kempe et al. in (2003) and also the others listed above primarily made assumptions about these probabilities. The methods used to assign influence probabilities/weight to edges include the following (Bonchi et al. 2011):

i. Using constant value for all edges (e.g. 0.1)

ii. Choosing value uniformly at random from a small set of constant. E.g. {.1, .25, .5}

iii. Using nodes in-degree.

To tackle this issue researchers recently attempt to take advantage of available information about users of online social networks performing some action, such as reading a blog, posting a picture, rating a movie etc. (Bonchi et al. 2011).

In section 2.4.1 we first discuss the structure of action log and definition of terms, such as *action propagation* and *user influence graph.* These terms are frequently used in the work done by Goyal et al. (2008) and Goyal et al. (2010). In section 2.4.2 we discuss frequent pattern approach by Goyal et al. (2008) to discover leaders (or influential nodes) from social network graph and its action log. And finally in section 2.4.3 we discuss the work of Goyal et al. (2010) where they used action log to trace action propagation to learn influence probability that can be used for influence maximization algorithms such as greedy of Kempe et al. (2003) and CELF of Leskovec et al. (2007).

2.5.1 Mining Action Log

Goyal et al. (2008) and (2010) presents techniques which takes Action Log, which is a relation Actions(User, Action, Time), along with a social graph $G(V, E)$ to mine for *influential* nodes (Goyal et al. 2008) and to learn *influential* probability (2010). Action log contains tuples, for example (u, a, t), which indicates that user u (such that $u \in V$) performed action a, at time t (Figure 17 b). This means the action log table contains such tuples for every action performed by every user of the social network. For example in Flixster (www.flixster.com), an online social network for movie rating, an action is a user rating a movie. In other words, if a user v rate "Mission Impossible" movie and later v's friend u does the same, then the action of rating the movie "Mission Impossible" propagates from user v to user u. Goyal et al. (2008) make use of such *propagation traces* available to tackle finding influential nodes or leaders from social graph (Discussed further in section 2.5.2). Also in (Goyal et al. 2010) they proposed to mine action log to compute influential probability, which we discuss in section 2.5.3.

Before we present these techniques we provide some important definitions. The projection of Actions on the first column is contained in the set of V of the social graph G. This means users in Actions table correspond to nodes of the social graph. Let us assume A be the universe of all actions. Using these Goyal et al. (2008 and 2010) defined propagation of action as follows:

> **Definition 6** ***Action Propagation*** **-** We say that an action $a \in A$ propagates from user v_i to v_j *iff* $(v_i, v_j) \in E$ and $\exists(v_i, a, t_i),(v_j, a, t_i) \in$ Actions with $t_i < t_j$. (Goyal et al. 2008)

Note that, for an action to propagate between v_i and v_j, there must be a social tie between v_i and v_j, both must have performed the action, one strictly before the other. Using the definition of action propagation we can define *Propagation Graph* as follows.

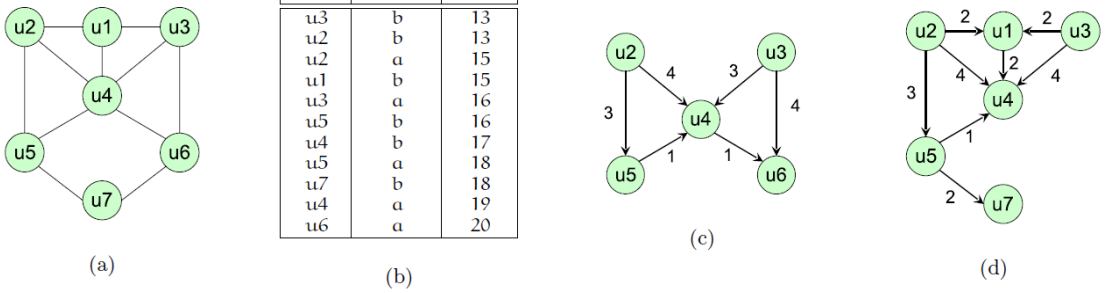


Figure 17 - (a) Example social graph; (b) A log of actions; (c) Propagation graph of action a and (d) of action b. (Goyal et al. 2008, Pg. 2)

> **Definition 7** ***Propagation Graph*** **-** For each action a, we define a propagation graph PG(a) = ($V(a)$, $E(a)$), defined as follows. $V(a)$ = {$v|\exists t$: (v, a, t) ∈ Actions}; there is a directed edge $v_i \xrightarrow{\Delta t} v_j$ whenever a propagates from v_i to v_j, with (v_i, a, t_i),(v_j, a, t_i) ∈ Actions, where $\Delta t = t_j - t_i$. (Goyal et al. 2008)

The propagation graph consists of users who performed the action and models the propagation of influence with edges connecting them in direction of propagation. Figure 17(c) and 17(d) is the Propagation Graph of action *a* and action *b* respectively.

2.5.2 Mining Leaders using Frequent Pattern approach

In Goyal et al. (2008) propose a solution based on the discovery of *frequent pattern of influence*, by mining the social graph and the action log. The goal is to identify the "leaders" in a social network. The general goal here is very similar to that of Kempe et al. (2003) and Leskovec et al. (2007), however the problem setting is different as they consider different set of input, such as action log, and applied pattern mining to discover leaders.

Motivated by frequent pattern mining and association rules mining, Goyal et al. (2008), define the notion of leadership based on how frequently a user exhibits influential behavior. In particular a user *u* is considered *leader* w.r.t. an action *a* provided a sufficient number of other users performed action *a* within a chosen time bound after *u* performed *a*. Furthermore these other users must be neighbour of *u* social network. If a user is found to act as a leader for sufficiently many actions, then it is considered a leader. Note based on this definition of leader we have three constraints which a leader must satisfy:

- For an action, should influence sufficiently large number of users ($>\psi$)
- For an action, should influence these users in a reasonable amount of time ($<\pi$)
- Should act as a leader in sufficiently large number of actions ($>\sigma$)

To aid in the definition of leaders, Goyal et al. (2008) defines the notion of an influence graph (Figure 18) next.

> **Definition 8** ***User influence graph*** **-** Given action *a*, a user *u*, and a maximum propagation time threshold π, we define the influence graph of the user *u*, denoted $Inf_{\pi}(u, a)$ as the subgraph PG(*a*) rooted at *u*, such that it consists of those nodes of PG(*a*) which are reachable from *u* in PG(*a*) and such that every path from *u* to any other node in $Inf_{\pi}(u, a)$ has an elapsed time at most π. (Goyal et al. 2008)

The elapsed time along a (directed) path in a propagation graph is the sum of edge labels along the path. E.g., in Figure 18(c), the elapsed time on the path u3→u5→u6 is 4 + 1 = 5.

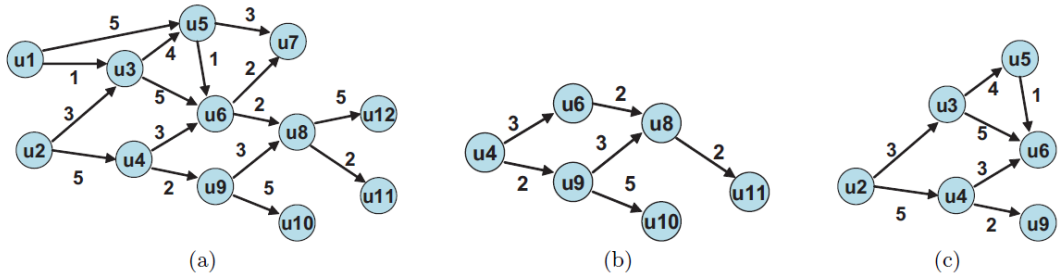


Figure 18 - The propagation graph of an action PG(a) in fig.(a), Inf_8(u4, a) in fig.(b), Inf_8(u2, a) in fig.(c). (Goyal et al. 2008, Pg. 4)

Figure 18 shows an example of propagation graph PG(*a*), and two user influence graphs. Based on this Goyal et al. (2008) formally defined the problem as follows:

> **Problem 3 (Leaders)** Given a set of actions I ⊆ A, and three thresholds π, ψ and σ, find all users $v \in V$ such that:
>
> $$\exists \boldsymbol{S} \subseteq \boldsymbol{I}, |\boldsymbol{S}| > \sigma: \forall a \in S. \text{size}\left(\text{Inf}_{\pi}(\boldsymbol{v}, \boldsymbol{a})\right) \geq \psi$$

Recall that a user to be selected as *leader,* based on above problem definition, he/she must perform number of actions larger than a given action threshold (σ). This is similar to *minimum frequency* constraint in pattern discovery and association rule (Goyal et al. 2008). To extract pattern of leaders based on the above definition Goyal et al. (2008) presented an algorithm (Algorithm 2 in figure 19) which scan the action log and computes an *influence matrix* $IM_{\pi}(U,A)$. Each entry of influence matrix, $IM_{\pi}(u,a)$, is the number of users performed the action *a* after *u* within time threshold π.

Example:

Once the above algorithm computes the influence matrix $IM_{\pi}(U,A)$ it can be used to compute leaders as defined in Problem 3. Given a user *u* and thresholds ψ (minimum number of user influenced for each action) and σ (minimum number of action to be influenced), let $L(u) = \{a| IM_{\pi}(U,A) > \psi \}$. That is L(*u*) is a set of all actions that user *u* influenced on at least ψ number of users. Now if |L(*u*)| > σ we can say node *u* is a leader. For example let us consider the following the influence matrix (table 6) computed by Algorithm 3 by taking social network graph and action log in figure 17 a and b.

Let us consider minimum number of user influenced for each action ψ to be 2 and minimum number of action to be influenced σ to be 1. For user u2, L(u2) = {a,b} because u2 influenced more than 2 users in both action *a* and *b.* Now |L(u2)| =2 which is greater than 1 (the value set for threshold σ). Therefore according to definition of leader as stated in Problem 3 we can conclude that user u2 is a leader. Similarly, L(u4)={} and therefore |L(u4)|=0 and that is why user u4 is not a leader.

Algorithm 2 Compute Influence Matrix

Input: Graph G; Action log Actions; Threshold π.

Output: Influence matrix IMπ(U,A).

1: Position a window of size π at the end of table Actions.

2: Discover the visible subgraph GW of G on the fly from the tuples in the window.

3: Compute the state of every node by starting from the most recent tuples and working backward up the graph.

4: Fill in the cells IMπ(u,a) whenever we have the state for node u w.r.t. action a computed.

5: **while** top of Actions not reached **do**

6: Move the window from the most recent tuple in the window to the next earlier tuple.

7: For every tuple that drops off the window, **update** the state of every other node.

8: For every new tuple that appears in the window, compute the state of the corresponding node by **propagating** the state from its children in the visible graph.

9: Update the IMπ entries as needed.

Figure 19: Compute Influence Matrix. (Goyal et al. 2008)

Table 6: Sample Influence Matrix

	a	**b**
u1	1	2
u2	3	4
u3	1	1
u4	0	1
u5	1	2
u6	1	2
u7	1	2

2.5.3 Learning Influence Probability

Goyal et al. (2010) tackles the problem learning influence probabilities among the users, by mining action log. To tackle this problem Goyal et al. (2010) introduces a framework in which they adopt an instance of *general threshold model* (Kempe et al 2003) which is defined as follows: Let us consider an inactive user *u* and the set of its active neighbors *S* (that is all nodes in *S* already performed certain task). To determine if *u* will activate (or perform the task), we first compute $p_u(S)$, which is the joint influence probability of *S* on *u*. If $p_u(S) \geq \theta_u$, where θ_u is the activation threshold of user *u*, according to GT model we conclude that *u* activates. Activation threshold is used to determine at what point a user will get activated. Goyal et al. (2010) defined the joint probability $p_u(S)$ as follows:

$$p_u(S) = 1 - \prod_{v \in S} (1 - p_{v,u})$$

To learn influence probability, $p_{v,u}$ for an edge $(v, u) \in E$, Goyal et al. (2010) first proposes 2 classes of influence probabilities models as follows:

> **Static Model** – In this model the influence probabilities are static and do not change over time.
>
> **Continuous Time (CT) Model** – In this model influence probabilities it changes over time.

Based on experiments and its results, it turns out that time-aware model are more accurate. However they are computationally very expensive to learn from large data sets. To tackle this Goyal et al. (2010) proposes an approximation of CT model called *Discrete Time model* which is defined as follows:

> **Discrete Time (DT) Model** – In this model the influence of an active user *v* on its neighbour *u* remains constant at $p(v,u)$ for a time window, $\tau_{v,u}$.

We limit our discussion to DT model as it is the most efficient and accurate as claimed by Goyal et al. (2010). To compute *p(u,v)* in DT model Goyal et al. (2010) introduces the notion of *credit*. When a user *u* performs an action under influence of its neighbours, who already performed the action before *u,* all the active neighbours of *u* are assigned or shares partial 'credit' for influencing *u* to perform the action. Let us suppose user *u* performs an action *a* and *S* is the set of its active users, then for each node $v \in S$ partial credit can be assigned using equation, where *I* is an indicator function and $t_u(a)$ is the time when *u* performs action *a.*:

$$credit_{v,u}(a) = \frac{1}{\sum_{w \in S} I(0 < t_u(a) - t_w(a) \leq \tau_{v,u})}$$

The indicator function *I* returns 1 for any node *v* if the user *u* performs an action *a,* within time $\tau_{v,u}$ and returns 0 otherwise. To compute $\tau_{v,u}$, the time window used in DT model, Goyal et al. (2010) suggests the following equation, where A_{v2u} is the number of action propagated from *v* to *u.*

$$\tau_{v,u} = \frac{\sum_{a \in A}(t_u(a) - t_v(a))}{A_{v2u}}$$

Using the above definition of *credit*, Goyal et al. (2010) proposes the following equation to compute *p(u,v)*, where A_v is the number of action performed by user *v.*

$$p(u, v) = \frac{\sum_{a \in A} credit_{v,u}(a)}{A_v}$$

To mine *p*(*u,v*) for static, CT and DT model Goyal et al. (2010) presents algorithms which scans action log and the social network graph (Figure 20 a and b). In online social

network the action log tend to be potentially huge with hundreds of thousands to millions of data. Due to this Goyal et al. (2010) pay particular attention in minimizing the number of scans of action log. They claim that their algorithm can learn the influence probability for all the models in no more than two scan of the action log.

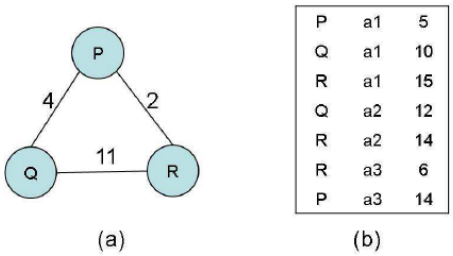
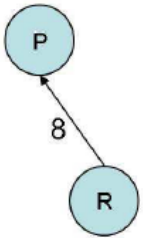


Figure 20: (a) Undirected social graph containing 3 nodes and 3 edges with timestamps when the social tie was created; (b) Action Log(User, Action, Time)
(Goyal et al. 2010. Pg. 6)

Example:

To show how $p(u,v)$ is computed in Discrete Time Model using method described above let us consider the social network graph and action log in figure 20 (Goyal et al. 2010). Figure 20(a) shows a social graph consisting of three nodes P, Q and R with three edges among them. Each edge is labeled with timestamps indicating at which time the two nodes became friends. The action log consists of 3 actions a1, a2 and a3 as shown in figure 20(b). First propagation graph is constructed for each action in the action log. Propagation graph for each action a1, a2 and a3 is in Figure 21 a, b and c respectively. Note that edges are directed in Propagation Graphs and labeled with time taken to propagate the action.

a) Propagation Graph for action a1 (b) Propagation Graph for action a2



(c) Propagation Graph for action a3

Figure 21: Propagation graphs of actions in action log of figure 20 b**.**

(Goyal et al. 2010. Pg. 6)

Using these propagation graphs we then need to compute the following:

A_{v2u} – Number of action propagated from *v* to *u.*

A_v – Number of action performed by *v.*

$\tau_{v,u}$ – Average time for user *u* to perform an action after *v* performs the same action.

$credit_{v,u}(a)$ – Total credit assigned to *v* for influencing *u.*

To show how we can compute all of the above parameters let us consider node P and Q, i.e. we like to compute influence probability of *p*(*P,Q*). A_{P2Q} is 1 as user P influences only one action, a1, on user Q according to propagation graphs (Figure 21). A_P is 2 as user P performs action a1 and a3. Similarly A_Q is also 2 as it performs action a1 and a2. Now using the formula for $\tau_{v,u}$,we can compute $\tau_{P,Q}$ as follows:

$$\tau_{P,Q} = \frac{\sum_{a \in A}\left(t_Q(a) - t_P(a)\right)}{A_{P2Q}}$$

$$= \frac{t_Q(a1) - t_P(a1)}{A_{P2Q}}$$

$$= \frac{10 - 5}{1}$$

$$= 5$$

To compute the credit given to user P for influencing Q for action a1 we use the following formula:

$$credit_{P,Q}(a1) = \frac{1}{\sum_{w \in S} I(0 < t_Q(a1) - t_w(a1) \leq \tau_{P,Q})}$$

$$= \frac{1}{\sum_{w \in S} I(0 < t_Q(a1) - t_w(a1) \leq 5)}$$

$$= \frac{1}{I(0 < t_Q(a1) - t_P(a1) \leq 5)}$$

$$= \frac{1}{1} = 1$$

Since no other neighbour of Q performed action a1 before Q other than user P, user P gets the full credit for influencing user Q for performing action a1. Based on these we then compute the influence probability *p*(P,Q), the probability of user P influencing user Q using the following:

$$p(P, Q) = \frac{\sum_{a \in A} credit_{P,Q}(a)}{A_P}$$

$$= \frac{credit_{P,Q}(a1)}{A_P}$$

$$= \frac{1}{2}$$

From the above we can conclude that the influence probability between users P and Q, is .5.

CHAPTER 3

PROPOSED MINING FOR INFLUENTIAL NODES FROM TRUST NETWORK

3.1 Trust-General Threshold Model

Note that the main problem we tackle in this thesis is to define and solve influence maximization considering both positive and negative influence among the nodes (or users) in trust network to find influential nodes. First we propose a new diffusion model, called Trust-General Threshold (TGT) Model, which is an extension of general threshold (GT) model by Kempe et al. (2003).

Recall that *general threshold model* (Kempe et al 2003) is a generalized model of LT and IC models which is defined as follows. Consider an inactive user *u* and the set of its active neighbours *S* (that is all nodes in *S* already performed a certain task). To predict whether *u* will activate (or perform the task), we need to determine $p_u(S)$, the joint influence probability of *S* on *u*. If $p_u(S) \geq \theta_u$, where θ_u is the activation threshold of user *u*, we can conclude according to GT model that *u* activates. Goyal et al. (2010) defined the joint probability$p_u(S)$ as follows, where $p_{v,u}$ is the influence probability of any node *v* on another node *u*:

$$p_u(S) = 1 - \prod_{v \in S}(1 - p_{v,u}) \qquad [1]$$

However in case of trust network the above joint probability equation is appropriate only if we consider only trusted neighbours of node *u.* Let us consider two scenarios for node C in figure 22. Let us assume node C trust node E, that is edge CE is +ve. Also node C distrust node A, that is edge CA is –ve. In the first scenario let us consider node E gets activated at time *t.* Also let us assume the influence probability $p_{E,C}$=0.3. So according to equation 1, the probability of node C getting activated (at time *t* + 1), is $p_C(\{E\}) = 1 - (1 - 0.3) = 0.3$. In the second scenario let us consider node E and A gets activated at

time *t.* Since node E is the only trusted user of node C the probability of node C getting activated in second scenario is also, i.e. $p_C(\{E\}) = 1 - (1 - 0.3) = 0.3$. In the second scenario we should also consider any negative influence on node C by node A, because C do not trust A. If few of trusted friends of a user adopt a product, the probability of him/her to also adopt a product should not stay same if few distrusted neighbours also adopt the product.

To accommodate such *negative* influence while computing $p_u(S)$, we introduce the notion of *negative influence probability*. Negative influence probability is the probability, denoted as $p^-_{v,u}$, of a user *u* not getting activated due to negative influence of user *v.* Here we assume that $p^-_{v,u} = 0$ if node *u* trusts node *v.* That is there is no negative influence on a user by any trusted neighbour. Similarly we also assume influence probability, from here on denoted as $p^+_{v,u}$, is 0 is node *u* not trust user *v.* That is there is no positive influence by any distrusted neighbours.

Let S^+ be all the trusted active neighbour of node *u* and S^- be the distrusted active neighbour of node *u.* Let us say $S = S^+ \cup S^-$.To predict whether *u* will activate given all nodes in *S* are active, we need to determine $p_u(S)$, the joint influence probability of *S* on *u*. To be able to compute $p_u(S)$, we first need to compute $p_u(S^+)$ and $p_u(S^-)$. To be able to do this we can simply use the equation 1 individually for S^+ and S^-. That is:

$$p_u(S^+) = 1 - \prod_{v \in S^+}\left(1 - p^+_{v,u}\right) \qquad [2]$$

$$p_u(S^-) = 1 - \prod_{v \in S^-}\left(1 - p^-_{v,u}\right) \qquad [3]$$

Note that $p_u(S^+)$ is the *positive* joint influence probability by all trusted neighbours of *u.* And $p_u(S^-)$ is the *negative* joint influence probability by distrusted neighbours of *u.* That is $p_u(S^-)$ is the probability of user *u* not performing the task or getting activated due to negative influence from S^-.

According to the proposed TGT model we say a node becomes active if:

$$p_u(S^+) > \theta^1 \text{ AND}$$

$$p_u(S^-) < \theta^2$$

Where θ^1 and θ^2 are thresholds that are chosen uniformly at random for each node. Thresholds, θ^1 and θ^2, essentially represent the latent tendencies of nodes to adopt a product when their trusted and distrusted neighbours do. The fact that these are randomly chosen is intended to model unavailability of these values in real life (Kleinberg 2007).

Example

Let us now demonstrate how we can compute $p_u(S)$ using the method proposed above. In figure 22 below, let us consider node C trusts node B and E. Also influence probabilities of node B and E on node C are 0.3 and 0.4 respectively. Furthermore let us also consider node A and D are not trusted by node C. Negative influence probabilities of node A and D on node C are 0.2 and 0.6 respectively. Also let us say threshold values θ^1 and θ^2 of node C are 0.5 and 0.7 respectively. Now let us say nodes A, B, D and E gets activated at the same time. To compute the probability of C also getting activated under trust-general threshold (TGT) model using equation 4, we first need to compute $p_C(S^+)$ and $p_C(S^-)$. Note that in this case $S^+ = \{B, E\}$ and $S^- = \{A, D\}$.

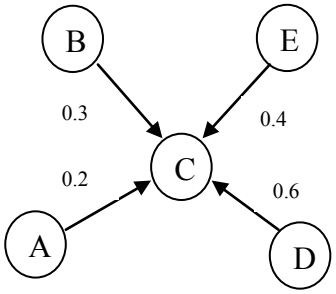


Figure 22: Example of social network graph with influence probability

We first use equation 2 to compute $p_C(S^+)$ as follows:

$$p_C(S^+) = 1 - \prod_{v \in S^+} \left(1 - p^+_{v,u}\right)$$

$$= 1 - \{\left(1 - p^+_{B,C}\right) * \left(1 - p^+_{E,C}\right)\}$$
$$= 1 - \{(1 - 0.3) * (1 - 0.4)\} = 0.58$$

That is the joint probability of node C to get activated by the influence of trusted neighbours of C is 0.58. Now we also need to consider any negative influence on C by distrusted neighbours, which are nodes A and D. Now we compute the joint probability, that is $p_C(S^-)$, of node C not getting activated under negative influence from its distrusted neighbours using equation 3.

$$p_C(S^-) = 1 - \prod_{v \in S^-} \left(1 - p^-_{v,u}\right)$$

$$= 1 - \{\left(1 - p^-_{A,C}\right) * \left(1 - p^-_{D,C}\right)\}$$
$$= 1 - \{(1 - 0.2) * (1 - 0.6)\} = 0.68$$

Note that $p_C(S^-)$ = 0.68 is the probability of node C *not* getting activated given node A and C gets activated and greater than $p_u(S^+) = 0.58$. And therefore under TGT model node C will get activated because $p_C(S^+) > \ \theta^1 = 0.5$ and $p_C(S^-) < \ \theta^2 = 0.7$.

Influence Maximization using the proposed Trust-General Threshold Model faces several challenges which we propose to tackle in this thesis. The first challenge we face while solving influence maximization in trust network is to tackle the problem of computing both positive and negative influence probabilities. To estimate these probabilities we use action log and extract pattern of user behavior. To do this we extract two types of frequent patterns from action log. The first pattern we extract, we call it Positive Frequent Action Pattern, is the number of actions performed by any user *u* after the same actions

were performed by a trusted neighbour of *u.* This is similar to extracting frequent item set in frequent pattern mining. In frequent pattern mining we are interested to find items which appear frequently in data. Similarly we propose to extract users who perform actions frequently after their trusted neighbours. Let us say a node u performs $A_{v.u}$ number of actions after its trusted neighbour *v,* and user *v* performs total of A_v tasks in total. We compute positive influence probability of node *v* on node *u* by using dividing $A_{v.u}$ by A_v. In frequent pattern mining this is also known as *confidence* which is interpreted as the probability of *u* performing an action after *v.* Note that unlike traditional frequent pattern mining we are also interested in computing number of times a user *do not* perform a task after its distrusted neighbour. Therefore we extract second pattern, we call it *Negative Frequent Action Pattern*, which counts the number of actions not performed by any user *u* after the same actions were performed by a distrusted neighbour of *u.* Let us say a node *u* do not performs $A'_{v.u}$ number of actions after its distrusted neighbour *v,* and user *v* performs total of A_v tasks in total. We compute negative influence probability of node *v* on node *u* by using dividing $A'_{v.u}$ by A_v. To illustrate this approach, let us consider that in a trust social network, a user *u* trust another user *v* and also distrust another user *w.* Now let us assume that according to action log user *v* performs in total of 3 actions. And out of these 3 actions 2 actions were performed by *u* after user *v* (trusted user of *u*) performs these same actions. So the probability of user *u* performing a task after user *v* performs the same action is 2/3 = 0.66. This is the positive influence probability of user *v* on user *u.* That is $p^+_{v,u}$ = 0.66. Based on this, for any nodes *u* and *v,* if $A_{v.u}$ is the number of actions performed by *u* after user *v* and A_v is the number of actions performed by user *v* positive influence probability of user *v* on user *u* can be expressed as:

$$p^+_{v,u} = \begin{cases} 0 \; if \; TM(u,v) is - ve \\ \dfrac{A_{v.u}}{A_v} \; otherwise \end{cases}$$

Now let us further consider that a user *w* (distrusted user of *u*) performs 4 tasks in total and out this only 1 task was performed by *u* after *w.* That is *u* did not perform 3 out of 4

tasks performed by *w*. So the negative influence probability of user *w* according Bernoulli distribution is 3/4 = 0.75. Based in this, for any nodes *u* and *v,* if $A'_{v.u}$ is the number of actions not performed by *u* after user *v* and A_v is the number of actions performed by user *v,* negative influence probability of user *v* on user *u* can be expressed as:

$$p^{-}_{v,u} = \begin{cases} 0 \; if \; TM(u,v) is + ve \\ \dfrac{A'_{v.u}}{A_v} \; otherwise \end{cases}$$

Note that to compute influence probabilities as discussed above, we need to learn required patterns of actions, such as $A_{v.u}$ and A_v from action log. We discuss these methods further in section 3.4.

Once both positive and negative influence probabilities are learned from action log, we are in position to discover influential nodes from trust network under TGT model. Recall *influence spread* (denoted as $\sigma(S)$) of any set of nodes *S* (*S* is a subset of *V* of social network graph *G*(*V,E*)) is the number of users (or nodes) getting activated (i.e. performing a task) given all the nodes in *S* is activated. Under linear threshold model and independent cascade model the influence spread function, $\sigma(.)$, is proves to be monotone and submodular (Kempe et al. 2003). Influence maximization, under these models, is basically to find a set *S* of maximum *k* nodes (*k* is given as input from end user) which is a subset of *V* of social network graph, *G*(*V,E*), such that influence spread, $\sigma(S)$, is maximized. This however is a NP-Hard problem (Kempe et al. 2003) and can be solved with Greedy (Kempe et al. 2003) or Lazy Forward optimization (Leskovec et al. 2007b) with 63% approximation guarantee. However these approaches rely on the fact that the influence spread function, $\sigma(.)$, is monotone and submodular. In contrast the influence spread function under the new proposed TGT model is non-monotone. That is adding a node (or user) may not result in influence spread to increase in TGT model. To show this let us consider the following scenario. Let *S* is the initially activated seed set and $\partial(S)$ are the nodes that were successfully activated by the seed set *S*. That is the influence spread, $\sigma(S)$, is actually the number of nodes in $\partial(S)$ or $|\partial(S)|$. Now let us consider a node *w* that has a negative influence (due to distrust) on two nodes *u* and *v* which is in $\partial(S)$. Now adding *w* to S will cause the probability of *u* and *v* getting activated to decrease according

to equation 4 and will not get activated. This will cause the influence spread of $S+w$, i.e. $\sigma(S+w)$, to decrease as $|\partial(S)-2| < |\partial(S)|$. Therefore we claim that influence spread function is not monotone. So to solve influence maximization under TGT model, the approximation guarantee by Greedy approaches by Kempe et al. (2003) and Leskovec et al. (2007) is not applicable. However we show that the spread function under TGT model is still sub modular. Therefore we define the problem of finding influential nodes from trust network as a problem of *maximizing non-monotone submodular function* under budget constraint. We propose to use approximate local search based algorithm which is based on (Lee et al. 2009) to solve influential maximization (that is to discover influential nodes), under TGT model, from a trust network subject to budget constraint. In the next section we discuss the overall framework of our proposed solution.

3.2 Solution framework

The overall solution framework, called Trust-Influential Node Miner (T-IM), for discovering influential node from trust network is illustrated in figure 23. Following are the input to T-IM framework:

i. Action Log – Contains tuples, for example <*a, u, t*>, which indicates that user *u* performed action *a*, at time *t.* (e.g. Figure 17 b).

ii. Trust Data – Contains tuples, for example <*u, v, trust*>. If trust is 1 it indicates that user *u* trust user *v*. If trust is 0 it indicates that user *u* does not trust user v.

iii. Budget – Number of influential nodes to be extracted.

The algorithm for TIM framework is listed in figure 24. The proposed solution consists of following four main steps listed below. Note that in rest of this thesis and in the algorithms presented, we denote adding an element v to any set S by $S = S + \{v\}$. Similarly we denote removing an element v from any set S by $S = S-\{v\}$. Also, V-S is the set of elements which are not in set S but in set V.

Step 1 - First the algorithm constructs a social network graph $G(V,E)$ using the Trust Table, T. For each tuple $(u,v,trust)$ it adds nodes v and u to set V of social network graph $G(V,E)$. It adds an edge (v,u) to set E of the social network graph. This is because if u trust (or distrust) v then there is an influence (positive or negative) of node v on node u.

In this step the trust matrix is also constructed by setting TM[u][v] to 1 or -1 based on the trust value of tuple (*u,v,trust*) in Trust Table T. Detailed steps of these processes using method called *Preprocess*() is given in figure 25 below.

Step 2 – In this second step the action log, L, is processed to compute parameters, $A_{v.u}$, $A'_{v.u}$ and A_v, which are required to compute the influence probabilities. We use frequent pattern based approach to extract these parameters from the Action Log, using an algorithm called Action Pattern Generator (APG) which we discuss in Section 3.3.

Step 3 – In this step influence probabilities are computed using method called *ComputeInfluence*(*u,v*), of each edge (*u,v*) in *E* of social network graph we constructed in Step 1 and stores it into Influence Matrix, IM[u][v]. The entry IM[u][v]of the influence matrix is the probability of user u influencing user v. IM[u][v]is positive influence probability if v trust u. Otherwise IM[u][v] is negative influence probability if v distrusts u. The *ComputeInfluence*(*u,v*) method is given in figure 26.

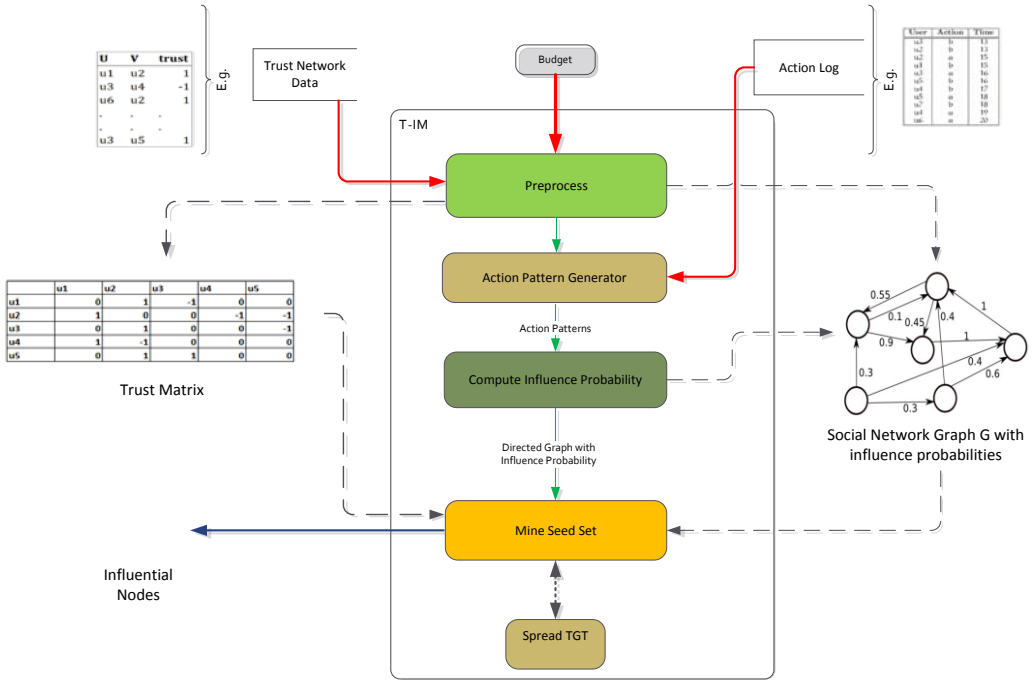


Figure 23: T-IM Framework

Algorithm: Trust based Influential Node Miner (T-IM)

Input:

1. ActionLog L // with tuples <*a, u, t*>
2. Trust Table T // with tuples <*u,v,trust*>
3. Budget // Number of nodes to be discovered

Output: Set of influential nodes (seed set), S

Other:

1. G(*V,E*) - Social Network Graph G where V is the set of all users and E is the set of directed edges.
2. TM[u][v] - TrustMatrix which is a 2D array where TM[u][v] is either 1, -1 or 0.
3. IM[u][v] - Influence Matrix which is a 2D array where IM[u][v] is positive (or negative) influence probability on user *v* by *u*.
4. A_v - Stores the number of action performed by a user *v*
5. $A_{v.u}$ - Stores number of action performed by *v* after trusted user *u*.
6. $A'_{v.u}$ - Stores number of action not performed by *v* after distrusted user *u*.

BEGIN

```
//Construct directed graph G(V,E) from T and Construct trust matrix
1. (TM, G)= Preprocess(T)
2. (A_{v.u}, A'_{v.u}, A_v )= APG(L,TM) //Compute action patterns
3. FOR each edge (u,v) in E
        {
            IM[u][v] = ComputeInfluence(u,v) //Compute influence probabilities
            IF IM[u][v] = 0
                E = E – {(u,v)} //Remove edge with 0 influence probability
        }
4. S=mineSeedLS(G(V,E),TM,IM, Budget) //Discover influential nodes
```

END

Figure 24: Trust based Influential Node Miner (T-IM) algorithm

Step 4 – This is the main and final step of our proposed framework which takes social network graph G(*V,E*), the trust matrix TM, and Influence Matrix IM to extract influential nodes. To mine influential nodes (or seed set) using local search based technique we use algorithm, called *mineSeedLS*(). Detail of this algorithm is given in section 3.4.

In section 3.5 we provide a running example of these steps using a simple data set.

```
Method: Preprocess(T)
Input: Trust Table T // with tuple <u,v,trust>
Output: G(V,E), TM
BEGIN
        E = NULL, V = NULL
        FOR each tuple <u,v,trust>  in T
                {
                        E = E + {(v,u)}
                        V = V + {v} + {u}
                        TM[u][v] = trust
                }
        Remove duplicates from V
        FOR each (u,v) not in E
             Set TM[v][u] = 0
END
```

Figure 25: The *Preprocess*() method.

Method: ComputeInfluence(*u,v*)
Input: Nodes *u* and *v*
Output: *p*(*u,v*) – The influence probability of node u on node v.

BEGIN
IF TM[*u*][*v*] = 1
p(u,v) = $\frac{A_{v.u}}{A_v}$
ELSE IF TM[*u*][*v*]= -1
p(*u,v*) = $\frac{A'_{v.u}}{A_v}$
ELSE *p*(*u,v*) = 0
RETURN *p*(*u,v*)
END

Figure 26: The *ComputeInfluence*(*u,v*) method.

3.3 Computing Positive and Negative Influence Probability

To learn influence probability, $p^{+}_{v,u}$ and $p^{-}_{v,u}$ in a trust network we propose to first mine for patterns of action performed (or not performed) by users in the network and compute required parameters, such as $A_{v.u}$ and A_v, from action log. Before we present the algorithm, which can compute these parameters, we provide some definitions first. We define *action sequence* of an action *a* as a sequence of users performing the action *a* with respect to time when it was performed. For example if an action *a* in action log is performed by user u1 followed by u2 and then u3, then action sequence *a,* denoted as Seq(*a*) = {u1, u2, u3}. Sub-sequence of any user *u* for action *a,* denoted as Seq(*a,u*) is the sequence of users performing action *a* before user *u.* For example sub-sequence of action *a* for user u1 in our previous example is Seq(*a,* u3) = {u2, u3}. We generate such sequence of all actions in action log by converting it to sequenced action log. An example of sequence action log of a corresponding action log is in figure 27 below.

Action	UserID	Time
a	u1	5
a	u2	6
a	u3	8
a	u4	11
b	u2	4
b	u3	5
b	u5	8
c	u2	11
c	u5	12
c	u1	18

a) Action Log sorted by Action and Time

Action ID	Seq(a)
a	u1, u2, u3, u4
b	u2, u3, u5
c	u2, u5, u1

b) Action Sequence Table

Figure 27: An example of Action Log and corresponding action sequence table.

Based on these we now present the algorithm, Action Pattern Generator (APG) that generates the patterns of action performed by users. The algorithm below (figure 28) takes the action log table and trust matrix as input and generates $A_{v.u}$ (# of actions performed by user *u* after trusted neighbour *v*), $A'_{v.u}$ (# of actions not performed by user *u* after distrusted neighbour *v*) and A_v (# of actions performed by user *u*) for all users *u* and *v* in the social network.

Algorithm: Action Pattern Generator (APG)

Input: Action Log Table (T), TrustMatrix (TM),

Output: $A_{v.u}$, $A'_{v.u}$and A_u for all users *u* and *v* in social network.

```
1.  BEGIN
2.  Generate sequence table ST from action log T.
3.  Set  A_{v.u} = 0 , A'_{v.u} = 0 and A_v = 0 for all users u and v
4.  FOR each action a in ST
5.      FOR each user v in action sequence Seq(a)
6.      {
7.          A_v += 1
8.          Su = SubSeq(a,v) // Get list of user performed the action a after user v
9.          Tu = Trust(v) // Get list of users who trust v
10.         // for each user u that trust u and performed action a after v
11.         FOR each user u in Su and Tu
12.           A_{v.u} ++ // Increment # of actions that user u influenced on user v
13.         Du = Distrust(v) // Get list of users who distrust v
14.         // for each user u that distrust v and did not perform the action a
15.         FOR each user v in Du and not in Su
16.           A'_{u.v} ++ // Increment # of actions that user v failed to influence on user v
17.     }
18. END
```

Figure 28: Algorithm: Action Pattern Generator (APG)

The algorithm first converts the action log table T to action sequence table, ST, as shown in figure 28 (Line 2). Then the algorithm initialize $A_{v.u}$, $A'_{v.u}$ and A_v to 0 (Line 3). Then for each user *v* in action sequence of *a,* that is in Seq(*a*), the algorithm will do the following. First it will increment A_v, the number of action performed by *v,* by 1 (Line 7). Then it will get list of user who performed the action *a* after the user *v*, that is Seq(*a,v*), and stores it to a list Su (Line 8). Also it will get list of users who trust *v* and stores it to list Tu (Line 9). This is done using a simple function trust(*v*) which returns a list of users who trusts *v.* Then for each user *u* that trust *v* and performed action *a* after *v*, that is in intersection of Su and Tu, the algorithm will increment $A_{v.u}$ by 1 (Line 11 and 12). After

this it will get list of users who do not trust *v* and stores it to list Du (Line 13). This is done using a simple function distrust(*v*) which returns a list of users who do not trust *v*. Then for each user *u* not in Su but in Du, we increment $A'_{v.u}$ (Line 15 and 16).

The above APG algorithm runs in $O(A*2N^2)$ in worst case, where *A* is the number of actions and *N* is number of nodes in *G*(*V,E*). This is because for action in action sequence table the algorithm will process each user who performs the action. Now in worst case each and every node performs each and every action. However in real life data this is not the case and indeed not realistic, so APG runs much faster than this.

3.4 Discovering Influential Nodes

Recall that to finding influential nodes is basically finding a set of nodes that would maximize the influence spread function, $\sigma()$. As highlighted before that the influence spread function that we need to maximize is not a monotone function. However we claim that though influence spread function in TGT model is non-monotone, but it is still sub modular. To show our claim we rely on the following theorem of Kleinberg (2007).

> **Theorem 3 (Kleinberg 2007):** For any instance of the General Threshold Model in which all the threshold functions are submodular, the resulting influence spread function, $\sigma()$, is submodular.

Note that the threshold functions in TGT model are basically equation 2 and 3. Here we show that equation 2, $p_u(S^+) = 1 - \prod_{v \in S^+}\left(1 - p^+_{v,u}\right)$, is sub modular.

Theorem 4: The joint influence probability $p_u(S^+) = 1 - \prod_{v \in S^+}\left(1 - p^+_{v,u}\right)$, of a node *u* by its trusted neighbour S^+ is submodular.

Proof (Goyal et. al 2010). Let S^+ be the set of trusted neighbours of *u* that are active and suppose a new trusted neighbour *w* of *u* gets activated. The new joint influence probability $p_u(S^+ + \{w\})$ can be computed incrementally from $p_u(S^+)$ as follows:

$$p_u(S^+ + \{w\}) = 1 - \prod_{v \in S^+} \left(1 - p^+_{v,u}\right)$$

$$= 1 - \left(1 - p^+_{w,u}\right) * \prod_{v \in S^+} \left(1 - p^+_{v,u}\right)$$

$$= 1 - \left(1 - p^+_{w,u}\right) * (1 - p_u(S^+))$$

$$= p_u(S^+) + \left(1 - p_u(S^+)\right) * p^+_{w,u} \quad \ldots\ldots\ldots [4]$$

Since individual probabilities are always between [0,1], $\left(1 - p_u(S^+)\right) * p^+_{w,u} > 0$.

Therefore $p_u(S^+) + \left(1 - p_u(S^+)\right) * p^+_{w,u} > \ p_u(S^+)$.

That is $p_u(S^+ + \{w\}) > \ p_u(S^+)$.

Therefore influence probability $p_u(S^+) = 1 - \prod_{v \in S^+}\left(1 - p^+_{v,u}\right)$ is monotone.

Now to show sub modularity let us first consider a set T^+ which is a subset of S^+. So according to property of sub modular function following must hold:

$$p_u(S^+ + \{w\}) - p_u(S^+) < \ p_u(T^+ + \{w\}) - p_u(T^+)$$

OR

$$p_u(S^+ + \{w\}) - p_u(S^+) - \ p_u(T^+ + \{w\}) + p_u(T^+) < 0$$

Lets evaluate the left hand side of the above inequality:

$$p_u(S^+ + \{w\}) - p_u(S^+) - \ p_u(T^+ + \{w\}) + p_u(T^+)$$
$$= \ \left(1 - p_u(S^+)\right) * p^+_{w,u} - \left(1 - p_u(T^+)\right) * p^+_{w,u} \text{ [According eq 4 above]}$$
$$= \ \left(p_u(T^+) - p_u(S^+)\right) * p^+_{w,u} < 0$$

As $p_u(T^+) < \ p_u(S^+)$ due to monotonocity. ■

Similarly we can also show that equation 3, $p_u(S^-) = 1 - \prod_{v \in S^-}\left(1 - p^-_{v,u}\right)$, is also sub modular. Based on these we claim that the approximation guarantee of (1-1/*e*) by greedy based approaches according to Theorem 1 of Nemhauser et al. (1978) is not applicable

for influence maximization under the new proposed TGT model. Recall that according to Nemhauser et al. (1978) the function that needs to be maximized must be monotone and submodular. So in contrast our problem of finding influential nodes from trust network is basically *maximizing a non-monotone submodular function subject to budget constraint k*. To solve this we propose an algorithm, mineSeedLS(), which is based on local search algorithm of Lee et al. (2009). MineSeedLS first picks a node, *v,* which achieves highest spread, $\sigma(v)$, according the TGT model and add it to the set of influential nodes, *S*. This step is same as the first step of the Greedy algorithm. Then we apply three local search based operations, namely *delete, add* and *swap* until we reach the budget *k.* Before we present the algorithm to find influential nodes under TGT model we first present the procedure to compute the influence spread, $\sigma()$. Recall influence spread of a given active seed set *S* is the total number of nodes activated after the diffusion process is over.

To compute total number of nodes activated we present algorithm, spreadTGT(), as shown in figure 29. The algorithm takes the following as input:

i. *Set of active set S*
ii. *Social Network Graph G(V,E)*
iii. *Trust Matrix TM*
iv. *Influence Matrix IM*

Recall entry of influence matrix, IM[u][v] is the positive influence probability ($p^{+}_{v,u}$) or negative influence probability ($p^{-}_{v,u}$), depending on whether *v* trusts *u* or not according to the trust matrix TM[u][v]. Note that each nodes *v* in the *V* of social network graph G is associated with the following node parameters:

i. *boolean v.active* [true or false. If true the node is active]
ii. *float v.inProbPos* [Joint positive influence probability $p_u(S^+)$]
iii. *float v.inProbNeg* [Joint negative influence probability $p_u(S^-)$]
iv. *float v.threshold1* [Positive threshold. Randomly assigned]
v. *float v.threshold2* [Negative threshold. Randomly assigned]

The output of the spreadTGT function is the number of users activated by initial active set *S* after the diffusion process, according to TGT model, is over. The spreadTGT() function starts with initializing the variable *spread* to size of the seed set S (Line 1). It also maintains a data structure queue T to store the list of nodes to be processed (Line 2). Then, for each active node *v* in seed set *S* (Line 3), the algorithm will process each neighbour *u* of *v* as follows (Line 4). First node *u* will be pushed to queue T if it is not already in T (Line 7). If TM[*u*][*v*] = 1 (Line 8) the algorithm will set *inProbPos* of node *u* to (1-IM[*v*][*u*]) and *inProbNeg* of *u* to 1, according to eq 2 and 3 (Line 9). Otherwise the algorithm will set *inProbPos* of node *u* to 1, and *inProbNeg* of *u* is set to 1 to (1-IM[*v*][*u*]) (Line 11). If node *u* was already in T the function will simply update the *inProbPos* and *inProbNeg* of node *u* according to equation 2 and 3.

Once all the neighbours of nodes *v* in seed set *S* is pushed to queue T, the algorithm calls another sub routine spreadTGT2() as shown in figure 30. The spreadTGT2 function takes queue T in addition to all the input that were passed to spreadTGT() function. Function spreadTGT2() actually traverse the social network graph to check which nodes in T meets the threshold requirement of TGT model (Line 4). If a node *u* in T meets the requirement the spreadTGT2() function activates the node *u* and then update the *spread* variable (Line 6). Then the function check for any neighbour *w* of *u*, not already in seed set *S*. Similar to function spreadTGT(), *inProbPos* and *inProbNeg* of node *w* is updated (Line 13 to 22). Then node *u* is removed from the queue T (Line 25). This process repeats until there are no more nodes getting activated in last iteration. Then the function returns variable *spread* to spreadTGT() function. Function spreadTGT() returns *spread* + *size of S* to its calling function (Line 22).

Algorithm: SpreadTGT (G(*V,E*),S,TM,IM)

//Estimates total number of nodes in G influenced by seed set S.

Input: Set of active set S; Social Network Graph G(*V,E*); Trust Matrix TM; Influence Matrix IM

Output: spread - # of users activate after the diffusion process is over

```
1. spread = |S| //Size of S
2. Queue T //T is a set of nodes that are to be processed
3. FOR each node v in S { //for loop 1
4.       FOR each neighbours u of v in G { //for loop 2
5.             IF u not in T //if node u is not processed yet.
6.              {
7.                    T.push(u)
8.                    IF TM[u][v]= 1
9.                          { u.inProbPos = (1-IM[v][u]);  u.inProbNeg = 1}
10.                   ELSE IF TM[u][v]= -1
11.                         { u.inProbNeg = (1- IM[v][u]);  u.inProbPos = 1 }
12.             }
13.           ELSE // Update inProbPos and inProbNeg of u
14.            {
15.                   IF TM[u][v]= 1
16.                         u.inProbPos = u.inProbPos * (1- IM[v][u]) //According to eq2
17.                   ELSE IF TM[u][v]= -1
18.                         u.inProbNeg = u.inProbNeg * (1- IM[v][u]) //eq3
19.           }
20.     } //end for loop 2
21. } // end for loop 1
22. RETURN spread+SpreadTGT2(T, G, S, TM, IM)
```

Figure 29: Algorithm: SpreadTGT (G(V,E),S,TM,IM)

Algorithm: SpreadTGT2 (T, G(*V,E*),S,TM,IM)

Input: T, G(*V,E*),S,TM,IM

Output: spread - # of users activate after the diffusion process is over

```
1. spread = 0; stop = false
2. WHILE stop  = false {
3.        stop = true; u = T.front()
4.        IF (1-u.inProbPos > u.Threshold1 && 1-u.inProbNeg<u.Threshold2)
5.        {
6.                u.active = true; spread++; stop=false;
7.                FOR each neighbours w of u      {
8.                        IF w is in S continue; //if w is in S do nothing.
19.                       ELSE  {
10.                               IF w not in T //if node u is not processed yet.
11.                               {
12.                                T.push(w)
13.                                IF TM[w][u]= 1
14.                                   { w.inProbPos = (1-IM[u][w]); w.inProbNeg = 1 }
15.                                ELSE IF TM[w][u]= -1
16.                                  { w.inProbNeg = (1-IM[u][w]);  w.inProbPos = 1 }
17.                               }
18.                               ELSE IF(w.active = false) {
19.                                       IF TM[w][v]= 1
20.                                               w.inProbPos = w.inProbPos * (1- IM[u][w])
21.                                       ELSE IF TM[u][v]= -1
22.                                               w.inProbNeg = w.inProbNeg * (1- IM[u][w])
23.                                   }
24.                   }// End for
25.    }// End if
26. T.pop() // } //End While
27. RETURN spread
```

Figure 30: Algorithm: SpreadTGT2 (T, G(V,E),S,TM,IM)

Now we present the algorithm which mines for influential nodes, called mineSeedLS (Figure 31). It takes social network graph G(*V,E*) and integer *budget*. The algorithm returns set of influential nodes, *S,* such that *S* is a subset of *V* and |S|<= *budget.* The algorithm starts by initializing seed set *S* to NULL (Line 1). Using spreadTGT() method discussed above the algorithm computes spread of each node *v* in *V*. The node with highest spread is picked and added to *S* (Line 3). Note that in the algorithm we denote adding a node *v* to a set S by $S = S+\{v\}$. Similarly we denote removing a node *v* from a set *S* by $S = S-\{v\}$. Also, *V-S* is the set of nodes which are not in set *S* but in set of all nodes *V*. The algorithm then performs the following local search operations:

Delete – If by removing any node *v* in *S* results in increasing the spread under TGT the node is removed from *S*. That is:

$$If\ v \in S, such\ that\ \boldsymbol{spreadTGT}(S-\{v\}) > \boldsymbol{spreadTGT}(S), then\ S = S-\{v\}$$

Add – If by adding any node *v* in *V-S* results in increasing the spread under TGT model the node is added to the set S. That is:

$$If\ v \in V-S, such\ that\ \boldsymbol{spreadTGT}(S+\{v\}) > \boldsymbol{spreadTGT}(S), then\ S = S+\{v\}$$

Swap- If by swapping any node *v* in S with any node *u* in *V-S* results in increasing the spread under TGT model the node *v* is removed from the set *S* and node *u* is added to the set *S*. That is:

$$If\ v \in S\ and\ u\ \in V-S, such\ that\ \boldsymbol{spreadTGT}(S-\{v\}+\{u\}) > \boldsymbol{spreadTGT}(S)$$
$$then\ S = S-\{v\}+\{u\}$$

If none of the above local search yields any further improvements in spread the algorithm stops and returns set of influential nodes *S*.

Algorithm: mineSeedLS - Mine influential nodes (seed set) under TGT using local search.
Input: Directed Graph *G(V,E)* and *budget*
Output: Set of influential nodes (seed set), S such that |S| <= Budget

```
BEGIN
1.      S = NULL
2.      v = argmax{spreadTGT(v) |v in V} //Pick v which yields maximum spreadTGT()
3.      S = {v}
4.      Boolean continue = true
5.      WHILE continue {
6.              continue = false
7.              FOR each v in S //Delete Operation
8.                      IF spreadTGT (S - {v}) > spreadTGT (S)
9.                      {
10.                             S = S - {v} // Delete v from S
11.                             continue = true
12.                             break //Break if at least 1 delete operation is done.
13.                     }
14.             IF (|S| <  budget)
15.               FOR each v in V //Add operation
16.                     IF spreadTGT (S + {v}) > spreadTGT (S)
17.                             {
18.                                     S = S + {v} // Add v
19.                                     continue = true
20.                                     break //Break if at least 1 add operation is done.
21.                             }
22.             FOR each v in S //Swap operaiton
23.                     FOR each u in V-S
24.                     IF spreadTGT (S - {v} + {u}) > spreadTGT (S)
25.                             {
26.                                     S = S - {v} + {u} // Swap node u and v.
27.                                     continue = true
28.                                     break //Break if atleast 1 swap operation is done.
29.                             }
30.                     END FOR
31.             END FOR
32.     END WHILE
33.     RETURN S
34. END
```

Figure 31: Algorithm mineSeedLS()

3.5 Complexity Analysis

The APG algorithm runs in $O(A*2N^2)$ in worst case, where A is the number of actions and N is number of nodes in $G(V,E)$. This is because for each action in action sequence table the algorithm will process each user who performs the action after any user v. Now in worst case each and every user performs each and every action. That means the algorithm will have to process N^2 number of times. The algorithm will also process each user who did not perform the action after any user v. That means the algorithm will have to process N^2 number of times. So for each action the algorithm have to process $N^2 + N^2 = 2N^2$ number of times. So if there are A number of actions in the action log the run time complexity of APG algorithm is $O(A*2N^2)$.

The local search based algorithm as defined in figure 31 could run for an exponential amount of time, until it reaches a locally optimal solution (Lee et al. 2009). To tackle this and to ensure polynomial time execution, we follow the standard approach (Lee et al. 2009) of an *approximate local search* under a suitable (small) parameter $\in > 0$. In approximate local search when looking for improvement, for example for adding a node, we multiply the spread of current seed set by a fraction $1+\in/n^4$ (where n is the size of the search space, that is number of nodes in this case).) For example when looking for local improvement for adding a node v with respect to current set S we will make following comparison:

$$\boldsymbol{spreadTGT}(S + \{v\}) > (1+\in/n^4)\boldsymbol{spreadTGT}(S)$$

3.6 Running Example

3.6.1 Example Dataset

To demonstrate the entire work flow of the Trust based Influence Maximization (T-IM) framework we will use a small sample dataset. Note that T-IM accepts three inputs namely, Trust Data table, Action Log table and an integer budget. The sample Trust Data is shown in table 7 and Action Log is shown in table 8. In this data set we have set of 10 users which are {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. In the trust data we have 21 trust relationships. The action log consists of 3 tasks, {A, B, C}, performed by users at various time. Note that action log table is sorted by action and time in ascending order. Also let us set our *budget* to 2, meaning we are looking for 2 influential nodes from this dataset.

Table 7: Example of Trust Data

User *u*	User *v*	trust
1	4	-1
1	7	1
2	1	1
2	3	1
2	8	-1
3	4	-1
3	10	-1
4	10	1
4	9	1
5	4	1
5	1	1
6	5	1
6	1	-1
7	6	1
7	8	-1
7	1	1
8	3	-1
8	9	1
9	2	1
9	10	1
10	3	-1

Table 8: Example of an Action Log

Action	User	Time
A	1	2
A	10	4
A	4	5
A	5	8
A	3	9
A	2	12
A	8	15
A	9	19
B	3	7
B	5	8
B	6	9
B	8	12
B	4	14
B	2	12
B	1	16
B	9	17
B	10	21
C	2	5
C	5	6
C	2	7
C	4	8
C	6	9
C	1	11
C	10	15
C	9	17
C	3	18
C	7	16

3.6.2 Preprocessing Step

We start with the preprocessing step which is the first step of T-IM. In this step a directed social network graph $G(V,E)$ is generated using the trust data in table 7. According to the *Preprocess*() method, as listed in figure 25, for each row (*u, v, trust*) in the trust data an edge (*v,u*) will be added to E of G. Also node u and v will be added to V of G. After processing all the row of the trust data any duplicate nodes in V will be removed. So the set of nodes, V, of social network graph $G(V,E)$ will be $V=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The graphical model of the social network graph $G(V,E)$ constructed using the trust data of table 7 is shown in figure 32. Note that the social network graph is directed where each edge (*v,u*) is added to it if there is a tuple (*u, v, trust*) in the trust data. For e.g. there is an edge (4,1) because in trust data table there is a row (1,4). Also note that the number of edges in the graph, which is 21, is exactly equal to number of rows in the trust data table.

Additionally in the preprocessing stage a trust matrix is constructed where each cell TM[u][v] of trust matrix will be set to the value of *trust* in tuple (*u, v, trust*). Any cell of trust matrix TM[u][v] is set to 0 if there is no trust/distrust relationship between u and v according to the trust table. The trust matrix constructed using the trust data of table 7 is shown in table 9 below. Note that TM[1][4] is -1 because there is a tuple (1, 4, -1) in the trust data table. Also TM[4][1] is 0 as there is no tuple where u is 4 and v is 1 in the trust data table.

Table 9: Trust Matrix

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	*0*	*0*	*0*	*-1*	*0*	*0*	*1*	*0*	*0*	*0*
2	*1*	*0*	*1*	*0*	*0*	*0*	*0*	*-1*	*0*	*0*
3	*0*	*0*	*0*	*-1*	*0*	*0*	*0*	*0*	*0*	*-1*
4	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
5	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
6	*-1*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
7	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*-1*	*0*	*0*
8	*0*	*0*	*-1*	*0*	*0*	*0*	*0*	*0*	*1*	*0*
9	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
10	*0*	*0*	*-1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

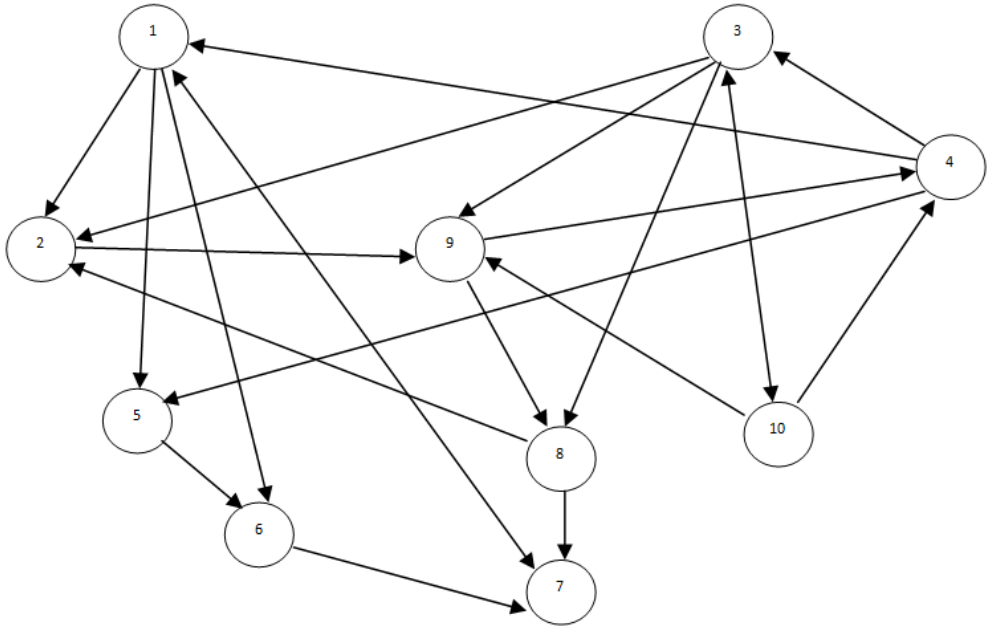


Figure 32: Social network graph modelled from trust data in table 7.

3.6.3 Computing Influence Probability using APG()

To compute influence probability of the each directed edge of social network work constructed in pre processing step earlier, we first use action pattern generator, APG(), as discussed in section 3.3. APG() algorithm (figure 28) is used to compute following patterns from action log (table 8):

- A_u - Stores the number of action performed by a user u
- $A_{v.u}$ - Stores number of action performed by v after trusted user u.
- $A'_{v.u}$ - Stores number of action not performed by v after distrusted user u.

First the APG() algorithm will convert the action log table to action sequence table as shown in table 10. Then for each action, let us say action A, the algorithm will process the above mentioned parameters for each user, u, who performed action A (that is in Seq(A)) as follows:

Step 1: Increment A_u by 1. For example, when processing the first row of the action table which is (A, 1, 2), the algorithm will update A_1 to 1. Values of A_u for all user v in G(V,E) according to action log in table 8 is shown in table 11.

Table 10: Action sequence table of action log in table 8.

Action	Action Sequence
A	1,10,4,5,3,2,8,9
B	3,5,6,8,4,2,1,9,10
C	2,5,2,4,6,1,10,9,3,7

Table 11: Values of A_u for all user *v* in G(*V,E*) according to action log in table 8.

U	A_u
1	3
2	3
3	3
4	3
5	3
6	2
7	1
8	2
9	3
10	3

Step 2: A list of users who performed the action *A* after user *u* (currently user 1), that is Seq(A,1) = {10, 4, 5, 3, 2, 8, 9}, is then generated and is stored to list Su.

Step 3: A list of users who trusts user *u* (currently user 1) is then extracted from the trust matrix and stored to list Tu. So T1 = {2, 5, 7}. For each user *v* in Su and Tu, that is who trusts user *u* and also performed action *A* after *u,* the parameter $A_{u.v}$ is incremented by 1. That is $A_{1.2}$ and $A_{1.5}$ is incremented by 1 because both node 2 and 5 trusts node 1 and also performed action A after their trusted neighbour node 1. Table 12 below lists all $A_{u.v}$ such that *v* trust *u.* Note that $A_{u.v}$ consists of 13 entries because there are 13 entries in the trust data table listed in table 8 which is trusted relationship (that is the trust value is 1).

Table 12: $A_{u.v}$ computed from action sequence table (table 9)

U	V	$A_{u.v}$
1	2	1
1	5	1
2	9	2
4	5	1
3	2	2
9	4	0
9	8	0
5	6	2
6	7	0
7	1	0
1	7	1
10	4	2
10	9	2

Step 4: A list of users who do not trusts user *v* is then extracted from the trust matrix and stored to list Du. So D1 = {8}. For all the users in Du and not in Su will be increment by 1. Node 8 in Du performed action *A,* therefore $A'_{1.8}$ will not be incremented. Table 13 below lists all $A'_{u.v}$ such that *v* do not trust *u.* Note that $A'_{u.v}$ consists of 8 entries because there are 8 entries in the trust data table listed in table 8 which is distrusted relationship (that is the trust value is -1).

Table 13: $A'_{u.v}$ computed from action sequence table (table 9)

U	V	$A_{u.v}$
1	6	1
4	1	1
4	3	1
3	10	2
8	2	2
8	7	2
3	8	1
10	3	1

Once the APG() algorithm process the action log and generates the required parameters as discussed above, TIM will then compute the influence probabilities of edge (*u,v*) of graph G(*V,E*) using *ComputeInfluence*() method listed in figure 26. For example for edge (1,2), the *ComputeInfluence*() method will check if node 2 trust 1 or not. According to TM[2][1] = 1, which means node 2 do trust node 1. Also according to $A_{u.v}$ in table 11, node 2 performed 1 action after node 1 (out of 3 actions in total by node 1). Therefore influence probability IM[1][2] = 1/3 = 0.333. Similarly influence probability of all edge in *E* is then computed and stored into the influence matrix as shown in table 14.

Table 14: Influence Matrix

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	0.33	0	0	0.33	0.33	0.33	0	0	0
2	0	0	0	0	0	0	0	0	0.67	0
3	0	0.67	0	0	0	0	0	0.33	0	0.67
4	0.33	0	0.33	0	0.33	0	0	0	0	0
5	0	0	0	0	0	0.67	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0.5	0.67	0	0	0	0	0.67	0

3.6.4 Mining Influential Nodes using MineSeedLS()

Before presenting the running example of MineSeedLS() algorithm using the trust matrix and influence matrix constructed above, we first present the running example how spreadTGT() method (figure 29) estimates the number of node getting activated by a given set of nodes. This method is repeatedly used by MineSeedLS(). Let us say we like to compute the spread, σ, of set of nodes S = {8 , 10} from our example. Also let's assume that for all nodes, both positive and negative threshold is set to 0.5.

Total spread of set S will be initialized to size of S which is 2. First the spreadTGT() method (figure 29) will create a queue T which is empty at this point. Then it will process neighbours of each node in S. So it will start with node 8 and it has two outgoing edges to nodes 2 and 7. Both 2 and 7 are not in T so it will added to T. Now for node 2 since TM[2][8] is -1, which means node 2 do not trust node 8 its *inProbNeg* (joint negative influence probability or $p_u(S^-)$) will be set to 1-IM[8][2] = 1 – 1 = 0. Similarly for node 7, since TM[7][8] is -1 its *inProbNeg* is set to 1 - IM[8][7] = 1 – 1 =0. The parameter *inProbPos* for both nodes 2 and 7 is set to 1. Using same steps neighbours of node 10 (the second node is set S) will be processed. Table 14 below list all the nodes in queue T after all the neighbours of initial seed nodes S is processed.

Table 15: Nodes with its joint influence probability in queue T

Node	inProbPos	inProbNeg	Active
2	0	0	False
7	0	0	False
3	1	1-0.5 =0.5	False
4	1-0.67=0.33	1	False
9	1-0.67=0.33	1	False

Once all the neighbours of initial seed set S is processed and stored into queue T, the spreadTGT() method will call spreadTGT2() (figure 30). A variable called *spread* is initialized to 0. In spreadTGT2() method, each element of T (table 15) will be evaluated to check if inProbNeg ($\boldsymbol{p_u(S^-)}$) and inProbPos ($\boldsymbol{p_u(S^+)}$) meets the threshold requirement of TGT model. Note that both positive and negative threshold for all nodes are set to 0.5. Only node 4 in T meets the threshold requirement as its (1-inProbPos) = 1-0.33 = 0.67 > 0.5 and (1-inProbNeg) = (1-1)=0 < 0.5. Due to the activation of node 4 the spread variable is incremented by 1, so the spread is now 1. Now spreadTGT2() will start

processing which additional nodes may get activated by new active node 4. It will get list of node 4's neighbours which are {1, 5, 3}. Nodes 1 and 5 are not in T, so these nodes will be added to T with their respective inProbNeg and inProbPos values. Node 3 is already in T, so its inProbNeg and inProbPos will simply updated according to equation 2 and 3. Since 3 do not trust 4 according to the trust matrix and IM[4][3] = 0.33, node 3's inProbNeg is now 0.5 * (1 – 0.33) = 0.335. Table 16 lists the updated queue T.

Table 16: Nodes with its joint influence probability in queue T

Node	inProbPos	inProbNeg	Active
2	0	0	False
7	0	0	False
3	1	1-0.5 =0.5	True
4	1-0.67=0.33	1	False
9	1-0.67=0.33	1	False
1	1	1-0.33 = 0.67	False
5	1-0.33=0.67	1	False

Then the spreadTGT() will further evaluate the nodes in T and check if any of non active nodes becomes further active. Only node 9 becomes as active according to TGT model as, 1 - inProbPos = 1 – 0.33 = 0.67 > 0.5 and 1 – inProbNeg = 1 – 1 = 0 < 0.5. Due to the activation of node 9 the spread variable is incremented by 1, so the spread is now 2. Now spreadTGT2() will start processing which additional nodes may get activated by new active node 9. It will get list of node 9's neighbours which are {8}. Note that node 8 is already in seed set S and is already active. Therefore there are no more nodes that may become active anymore. The process stops at this point and returns the value of spread which is 2 to the calling function spreadTGT(). The spreadTGT() will then return 2 + 2=4, because the size of the initial active nodes S is 2 and the spreadTGT2() returns 2.

Now let us show the how mineSeedLS() algorithm finds influential nodes using spreadTGT() method we discussed above. First the algorithm will compute spread of each node as singleton using spreadTGT(). That is for every node v in V the algorithm will compute spreadTGT($\{v\}$). The node with maximum spread will be picked and stored into the set S of mineSeedLS(). By running the spreadTGT() for each node we will get the spread as shown in table 17. As node 3 have the highest spread it will picked and added to set of influential nodes S, which is now {3}. Note that node 10 have also spread 3 but was processed after 3, which is why node 3 is picked instead of node 10. Now the

local search is going to start. The delete operation is skipped as there is only 1 node is set *S*. Since our budget is 2 the algorithm continues to see if adding any node results in improving the spread. It picks node 1 as spreadTGT({3} + 1) = 4 > 3 (spreadTGT({3} = 3). So the set of influential set have two nodes {3, 1}. Note that the spread of current set S is now 4. Then it continues to check if swapping (or exchanging) any node in *S* with any node in *V* (but not already in S) yields any improvement in spread. The spread of set *S* as a result of removing node 1 and replacing it with node 5 is 5, that is spreadTGT({3,1} – 1 + 5) = 5. This is actually an improvement from previous spread of 4. So node 1 is dropped and node 5 is added to set *S*, which now is {3, 5}. The algorithm will continue search for any further improvement. First it checks if dropping any element from *S* improves the spread or not. Since spreadTGT({3,5} – 3) =3 and spreadTGT({3,5} – 3) =2 and do not improve the previous spread of 5, no element is dropped. Also the size of S is now 2 which is our budget the algorithm will not look for adding any new node. It will again check if swapping (or exchanging) any node in *S* with any node in *V* (but not already in *S*) yields any improvement in spread. No exchange yields any further improvement from pervious spread of 5. So the algorithm stops at this point and return set *S* = {3, 5} which manages to achieve total gross spread of 5.

Table 17: Spread of each node

Node	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Spread	1	2	3	1	2	1	1	1	1	3

CHAPTER 4

EXPERIMETNS AND ANALYSIS

4.1 Dataset

4.1.1 Epinions Dataset

Epinions.com is a general consumer review site where visitors can read reviews about a variety of items to help them decide on a purchase or they can join for free and begin writing reviews. Users of the Epinions.com can declare whether to "trust" or “distrust” each other.

Table 18: Epinions trust dataset

Column Name	Description
MY_ID	This stores Id of the member who is making the trust/distrust statement.
OTHER_ID	The other ID is the ID of the member being trusted/distrusted
VALUE	Value = 1 for trust and -1 for distrust
CREATION	It is the date on which the trust was made

In this research we use Epinions dataset, provided by Masa and Avesani (2006) and downloaded from http://www.trustlet.org that has two types of information. The first dataset consists of trust and distrust information which have fields as listed in table 18. Each row in this data represents a link between MY_ID (u) and OTHER_ID (v). The VALUE field is either 1, meaning u trust v, or -1, meaning u do not trust v. In this dataset we identified about 95, 318 nodes with 11,56,753 edges. Out of these about 86% of the edges are positive and 14% is negative. However since our main goal is to show quality of nodes selected my mineSeedLS is better than that of greedy based solution under TGT model and also any network with nodes more than 10,000 may run for days, we select a small snap shot from the dataset comprising of approximately 10,000 nodes.

The second dataset consists of rating information (table 19). Epinions users can post review on any certain product. Other users can rate these reviews from 5 (“Very Helpful") to 1 (“Not Helpful") etc. We use this dataset to extract action log of users in the network. Whenever a user rates a review it is considered as an action performed by the users. This dataset consists of 13,668,319 article ratings. We use this dataset as our

'Action Log'. Rating of each object is first classified as 'High' (if the rating is between 3-5) and 'Low' (if the rating is between 1-2). We consider two users perform same action if they rated a same object as 'High' or 'Low'.

Table 19: Epinions rating dataset

Column Name	Description
OBJECT_ID	The object ID is the object that is being rated.
MEMBER_ID	Stores the id of the member who is rating the object
RATING	Stores the rating value between 1-5.
CREATION	The date on which the member first rated this object

4.1.2 Wikipedia Dataset

Wikipedia is a very popular free online encyclopaedia. And many volunteers around the world contribute to maintain and write articles on different topics. A small part of such contributors are 'administrators', who have access to additional technical features that helps in maintenance. Users of Wikipedia can vote for or against another user to become administrator. The data set we collected from 'The Koblenz Network Collection' (http://konect.uni-koblenz.de/) consists of network of users from the English Wikipedia that voted for and against each other in admin elections. Nodes represent individual users of Wikipedia. And edges represent votes, which can be positive ("for" vote) and negative ("against" vote). In the dataset we have about 8,297 nodes and about 107,071 edges. Unfortunately there was no 'Action Log' available for this dataset. So we assigned influence probability uniformly and randomly to each edge.

4.2 Performance Analysis

The goal of our experiments is to show that influence spread achieved by our MineSeedLS algorithm improves influence spreads that can be achieved by standard approaches like CELF of Leskovec et al. (2007). We compared influence spread, number of nodes activated by seed set discovered, achieved by our proposed T-IM framework with the following approaches:

CELF-TGT: This is the greedy algorithm of [6] with the CELF optimization [10].

Degree-TGT: For comparison, we also compare our approach with a simple heuristic that selects the top k vertices with the highest degrees [6] [2]. Since we are dealing with trust network we select vertices with largest positive in degree.

Figure 33 show the influence spreads of various algorithms on trust network graph generated from Wikipedia dataset. Our T-IM performs very closely to CELFgreedy for smaller seed sets (<10). However it outperforms CELFGreedy for seed set size > 15. It also outperforms DegreeHeuristic for all seed set sizes. Also in Epinions dataset the spread achieved by T-IM outperforms both Degree and CELF based solutions (Figure 34). As expected CELF performs inconsistently in both datasets and in some cases it even performs below the Degree based solution. Recall that mineSeedLS performs additional operations such as *delete* and *swap* in attempt to improve the spread of selected set of influential nodes. On the other hand the Greedy (Kempe et al. 2003) just keep on adding nodes to the set of influential nodes based of maximum marginal gain. Due to these additional operations of mineSeedLS, it outperforms Greedy (Kempe et al. 2003) in terms of influence spread under the new proposed TGT model. Note that in Greedy (Kempe et al. 2003) algorithm gives good solution only when the influence function is monotone. Monotone property ensures that, adding a node will always increase the spread, which is why we do not need to track back and re-evaluate the solution. However in the case of influence spread under TGT models we the function is non-monotone. That is adding a node does not always guarantee an increase in spread, in fact spread may decrease. So we require tracking back and re evaluate our solution, by for example removing or swapping nodes. This actually ensures the quality of seed by such local search operations is better than that of greedy in case of TGT model. Our experimental results also validate this claim.

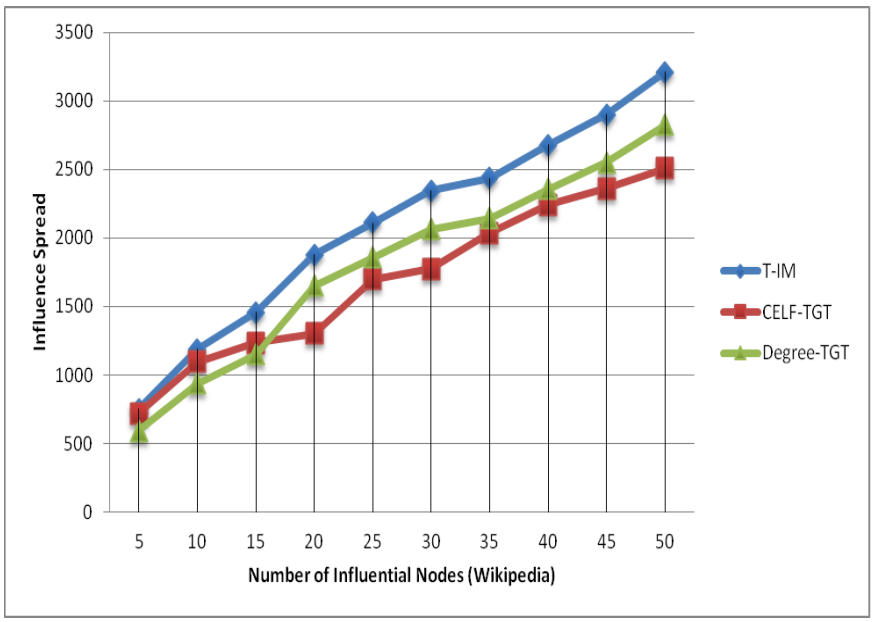


Figure 33: Influence spreads of different algorithms on Wikipedia Dataset under TGT model

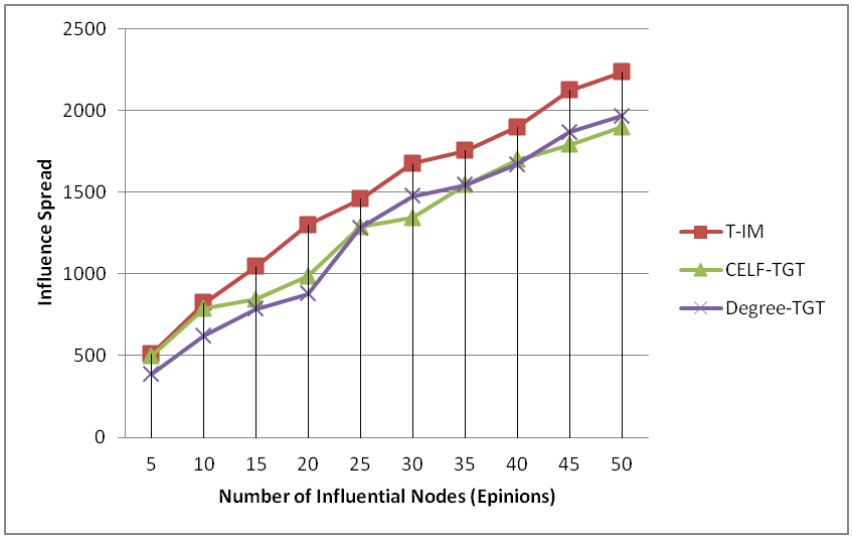


Figure 34: Influence spreads of different algorithms on Epinions Dataset under TGT model

4.3 Runtime Analysis

4.3.1 Runtime of APG

Figure 35 shows the scalability of APG algorithm. The runtime is almost linear function to the number of the records read from action log.

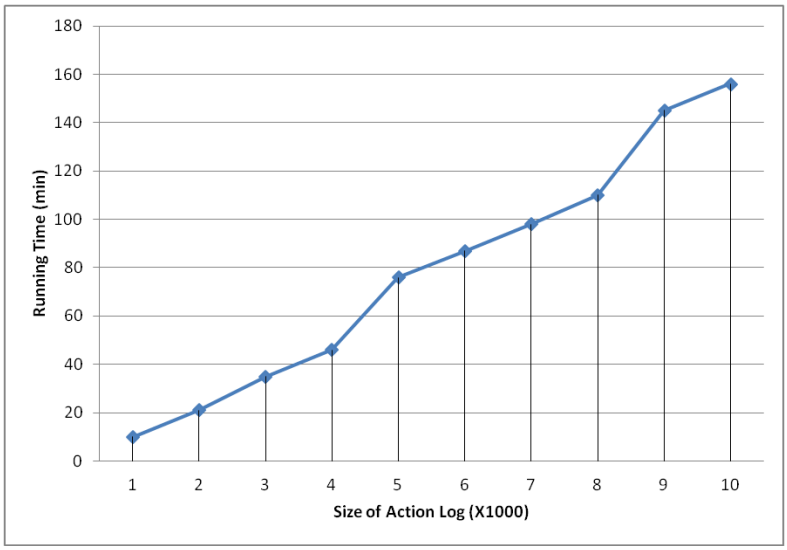


Figure 35: Runtime of APG with various size of Action Log

4.3.2 Runtime of mineSeedLS

To compare runtime of mineSeedLS with CELF and Degree based heuristic we recorded time required to select influential nodes of different sizes. Figure 36 reports the runtime comparison on Epinions dataset and figure 37 reports the same on Wikipedia dataset. In both datasets Degree heuristic performs almost in constant time. MineSeedLS takes longer than CELF as the size of the required set of influential nodes increases in both datasets. This was expected as mineSeedLS performs additional operations such as *delete* and *swaps* which is computationally very expensive. For example to perform a *swap* operation the algorithm requires to remove each element in seed set S with every element not in S but in V. This shows room for improvement of mineSeedLS in terms of scalability. As mentioned earlier, scalability was not focus of this thesis; however there are several ways to make the approach more scalable. We discuss some of these approaches in the next chapter.

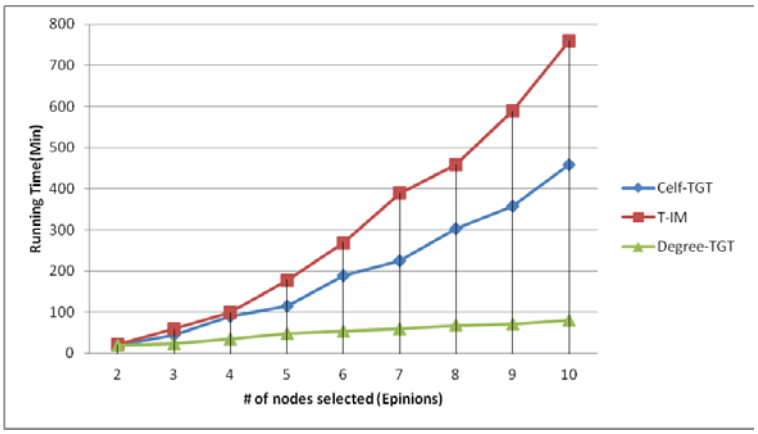


Figure 36: Running time of different algorithms on Epinions Dataset under TGT model

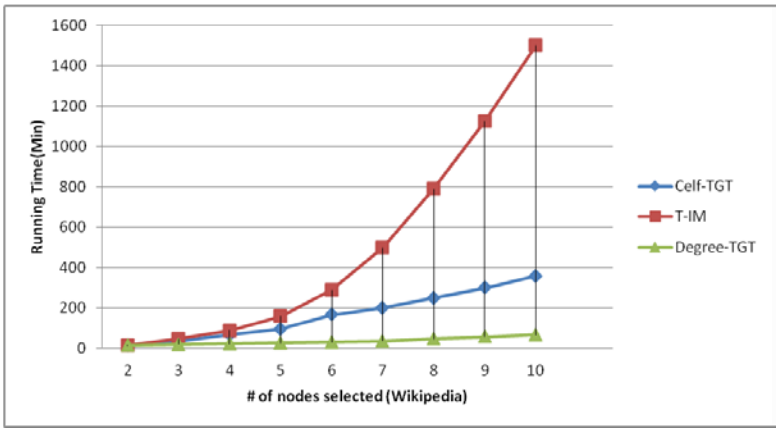


Figure 37: Running time of different algorithms on Wikipedia Dataset under TGT model

CHAPTER 5

CONCLUSIONS AND FUTURE WORKS

Analyzing information diffusion and social influence in social networks has various real-world applications. Influence maximization (IM) in viral marketing is an example of such an important application. In this research we tackled the influence maximization problem in trust network. We argue that to find influential nodes from a trust network we need to model the diffusion process by considering both positive and negative influence exerted by trusted and distrusted neighbours. Motivated by this we introduce a new diffusion model, called Trust-General Threshold (TGT) model, where both positive and negative influence exists. We showed that unlike existing diffusion models, influence maximization under proposed TGT model is a problem of *maximizing non-monotone sub-modular function*.

To learn influence probabilities, which are required parameters for TGT model, we propose an algorithm, Action Pattern Generator (APG), to mine action logs to extract frequent patterns of actions performed by users in trust network. Using this we estimate both positive and negative influence probabilities required for the TGT model. Then we propose an algorithm, called mineSeedLS, using local search technique (Lee et al. 2009) to find influential nodes. We ran experiments on real life dataset collected from Epinions and Wikipedia to show that quality of nodes selected by our proposed mineSeedLS outperforms existing benchmark algorithms such as CELF(Leskovec et al. 2007b) by almost 35%.

However as expected the scalability of minedSeedLS is not suitable for large social network. Previously scalability is tackled in Influence Maximization under various models such as LT and IC (Chen et al. 2009). In future we want to adopt some these methods in our TGT model to make it more scalable. Also we have plans to use hybrid approaches that combine the advantages of different algorithms, such as clustering and community detection, to improve the efficiency and effectiveness of influence maximization under TGT model. Further more in future we wish to tackle influence maximization considering dynamic network. One key property of any social network is that it is changing all the time, especially in online domain.

BIBLIOGRAPHY

1. Charu C. Aggarwal. 2005. On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st international conference on Very large data bases.* VLDB '05. VLDB Endowment 901-909.
2. Agrawal R., Imielinski T., Swami A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Washington D.C., USA, 207-216.
3. Agrawal R., Srikant R. 1994. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th VLDB conference*, Santiago, Chile, 487-499.
4. Agarwal, N., Liu, H., Tang, L., and Yu, P. S. 2008. Identifying the influential bloggers in a community. In *Proceedings of the international conference on Web search and web data mining*. WSDM ’08. ACM, New York, NY, USA, 207–218.
5. Berger-Wolf, T. Y. and Saia, J. 2006. A framework for analysis of dynamic social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* KDD ’06. ACM, New York, NY, USA, 523–528.
6. Bhagat, S., Cormode, G. and Muthukrishnan, S. 2011. Node classification in social networks. In *C. Aggarwal, editor, Social Network Data Analytics*. Springer, 1 edition, 115–148.
7. Bonchi, F., Castillo, C., Gionis, A., and Jaimes, A. 2011. Social network analysis and mining for business applications. *ACM Trans. Intell. Syst*. Technol. 2, 3 (May), 22:1–22:37.
8. Cha, M., Mislove, A., and Gummadi, K. P. 2009. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th international conference on World wide web*. WWW ’09. ACM, New York, NY, USA, 721–730.
9. Chen, W., Wang, C., and Wang, Y. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining.* KDD ’10. ACM, New York, NY, USA, 1029–1038.

10. Chen, W., Wang, Y., and Yang, S. 2009. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* KDD ’09. ACM, New York, NY, USA, 199–208.
11. Chen, W., Yuan, Y., and Zhang, L. 2010. Scalable Influence Maximization in Social Networks under the Linear Threshold Model. In *Proceedings of the 2010 IEEE International Conference on Data Mining* (ICDM '10). IEEE Computer Society, Washington, DC, USA, 88-97.
12. Domingos, P. and Richardson, M. 2001. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* KDD ’01. ACM, New York, NY, USA, 57–66.
13. Easley, D. and Kleinberg, J. 2010. Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press.
14. Feige, U.; Mirrokni, V.S.; Vondrak, J. 2007. Maximizing Non-Monotone Submodular Functions. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science* (FOCS '07). IEEE Computer Society, Washington, DC, USA, 461-471.
15. Getoor, L. 2003. Link mining: a new data mining challenge. *SIGKDD Explor*. Newsl. 5, 84–89.
16. Gomez Rodriguez, M., Leskovec, J., and Krause, A. 2010. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’10. ACM, New York, NY, USA, 1019–1028.
17. Goyal, A., Bonchi, F., and Lakshmanan, L. V. 2008. Discovering leaders from community actions. In *Proceeding of the 17th ACM conference on Information and knowledge management.* CIKM ’08. ACM, New York, NY, USA, 499–508.
18. Goyal, A., Bonchi, F., and Lakshmanan, L. V. 2010. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining.* WSDM ’10. ACM, New York, NY, USA, 241–250.
19. Goyal, A., Bonchi, F., and Lakshmanan, L. V. S. 2011. A data-based approach to social influence maximization. *Proc. VLDB Endow*. 5, 73–84.

20. Goyal, A., Lu, W., and Lakshmanan, L. V. 2011. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web.* WWW ’11. ACM, New York, NY, USA, 47–48.
21. Gruhl, D., Guha, R., Liben-Nowell, D., and Tomkins, A. 2004. Information diffusion through blogspace. In *Proceedings of the 13th international conference on World Wide Web.* WWW ’04.ACM, New York, NY, USA, 491–501.
22. Guha, R., Kumar, R., Raghavan, P., and Tomkins, A. 2004. Propagation of trust and distrust. *ACM Press*, 403–412.
23. Hartline, J., Mirrokni, V., and Sundararajan, M. 2008. Optimal marketing strategies over social networks. In *Proceeding of the 17th international conference on World Wide Web*. WWW ’08. ACM, New York, NY, USA, 189–198.
24. Karamon, J., Matsuo, Y., and Ishizuka, M. 2008. Generating useful network-based features for analyzing social networks. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*. AAAI’08. AAAI Press, 1162–1168.
25. Kempe, D., Kleinberg, J., and Tardos, E. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’03. ACM, New York, NY, USA, 137–146.
26. Kimura, M. and Saito, K. 2006. Tractable models for information diffusion in social networks. In *Knowledge Discovery in Databases: PKDD 2006*. Lecture Notes in Computer Science, vol. 4213. Springer Berlin / Heidelberg, 259–271. 10.1007/11871637.
27. Kimura, M., Saito, K., Nakano, R., and Motoda, H. 2009. Finding influential nodes in a social network from information diffusion data. In *Social Computing and Behavioral Modeling*. Springer US, 1–8. 10.1007/978-1-4419-0056-218.
28. Kleinberg, J. M. Challenges in mining social network data: processes, privacy, and paradoxes. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’07. ACM, New York, NY, USA, 4–5.
29. Kleinberg, J. 2007. Cascading behavior in networks: algorithmic and economic issues. Cambridge University Press.

30. Kunegis, J., Lommatzsch, A., and Bauckhage, C. 2009. The slashdot zoo: mining a social network with negative edges. In *Proceedings of the 18th international conference on World wide web*. WWW ’09. ACM, New York, NY, USA, 741–750.
31. Lee, J., Mirrokni, V. S., Nagarajan, V., and Sviridenko, M. 2009. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proceedings of the 41st annual ACM symposium on Theory of computing* (STOC '09). ACM, New York, NY, USA, 323-332.
32. Leskovec, J., Adamic, L. A., and Huberman, B. A. The dynamics of viral marketing. 2007a. *ACM Trans*. Web 1.
33. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. 2007b. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’07. ACM, New York, NY, USA, 420–429.
34. Leskovec, J., Huttenlocher, D., and Kleinberg, J. 2010a. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*. WWW ’10. ACM, New York, NY, USA, 641–650.
35. Leskovec, J., Huttenlocher, D., and Kleinberg, J. 2010b. Signed networks in social media. In *Proceedings of the 28th international conference on Human factors in computing systems*. CHI ’10. ACM, New York, NY, USA, 1361–1370.
36. Liben-Nowell, D. and Kleinberg, J. 2003. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*. CIKM ’03. ACM, New York, NY, USA, 556–559.
37. Massa, P. and Avesani, P. Trust metrics in recommender systems. Massa, P. and Avesani, P. 2007. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems.* RecSys ’07. ACM, New York, NY, USA, 17–24.
38. Massa, P., & Avesani, P. 2006. Trust-aware bootstrapping of recommender systems. In *Proceedings of ECAI 2006 Workshop on Recommender Systems* (pp. 29-33).
39. Mathioudakis, M., Bonchi, F., Castillo, C., Gionis, A., and Ukkonen, A. 2011. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’11. ACM,

New York, NY, USA, 529–537.

40. Mossel, E. and Roch, S. 2007. On the submodularity of influence in social networks. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. STOC ’07. ACM, New York, NY, USA, 128–134.
41. Provost, F., Dalessandro, B., Hook, R., Zhang, X., and Murray, A. 2009. Audience selection for on-line brand advertising: privacy-friendly social network targeting. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’09. ACM, New York, NY, USA, 707–716.
42. Richardson, M. and Domingos, P. 2002. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’02. ACM, New York, NY, USA, 61–70.
43. Saito, K., Kimura, M., Ohara, K., and Motoda, H. 2010. Behavioral analyses of information diffusion models by observed data of social network. In *Advances in Social Computing, S.-K. Chai, J. Salerno, and P. Mabry, Eds. Lecture Notes in Computer Science.* Vol. 6007. Springer Berlin.
44. Saito, K., Nakano, R., and Kimura, M. 2008. Prediction of information diffusion probabilities for independent cascade model. In *Proceedings of the 12th international conference on Knowledge- Based Intelligent Information and Engineering Systems, Part III*. KES ’08. Springer-Verlag, Berlin, Heidelberg, 67–75.
45. Senator, T. E. 2005. Link mining applications: progress and challenges. *SIGKDD Explor. Newsl.* 7, 76–83.
46. Song, X., Chi, Y., Hino, K., and Tseng, B. 2007a. Identifying opinion leaders in the blogosphere. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management.* CIKM ’07. ACM, New York, NY, USA, 971–974.
47. Staab, S., Domingos, P., Mike, P., Golbeck, J., Ding, L., Finin, T., Joshi, A., Nowak, A., and Vallacher, R. 2005. Social networks applied. *Intelligent Systems, IEEE 20*, 1 (jan.-feb.), 80 – 93.
48. Tang, J., Sun, J.,Wang, C., and Yang, Z. 2009. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on*

Knowledge discovery and data mining. KDD ’09. ACM, New York, NY, USA, 807–816.

49. Tantipathananandh, C., Berger-Wolf, T., and Kempe, D. 2007. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’07. ACM, New York, NY, USA, 717–726.
50. Wang, J., Luo, Y., Zhao, Y., and Le, J. 2009. A survey on privacy preserving data mining. In *Database Technology and Applications, 2009 First International Workshop on*. 111 –114.
51. Wang, Y., Cong, G., Song, G., and Xie, K. 2010. Community-based greedy algorithm for mining top-*k* influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. KDD ’10.* ACM, New York, NY, USA, 1039–1048.
52. Wellman, B. & Berkowitz, S. 1988. Social Structures: A Network Approach, Cambridge University Press .

VITA AUCTORIS

Sabbir Ahmed was born in 1979 in Dhaka, Bangladesh. He received his Bachelors degree in Computer Science from University of Windsor, Windsor, Ontario in 2002. His research interests include data mining, social network analysis and machine learning.