8-2014

REPUTATION COMPUTATION IN SOCIAL NETWORKS AND ITS APPLICATIONS

JooYoung Lee
Syracuse University

ABSTRACT

This thesis focuses on a quantification of reputation and presents models which compute reputation within networked environments. Reputation manifests past behaviors of users and helps others to predict behaviors of users and therefore reduce risks in future interactions. There are two approaches in computing reputation on networks- namely, the macro-level approach and the micro-level approach. A macro-level assumes that there exists a computing entity outside of a given network who can observe the entire network including degree distributions and relationships among nodes. In a micro-level approach, the entity is one of the nodes in a network and therefore can only observe the information local to itself, such as its own neighbors behaviors. In particular, we study reputation computation algorithms in online distributed environments such as social networks and develop reputation computation algorithms to address limitations of existing models. We analyze and discuss some properties of reputation values of a large number of agents including power-law distribution and their diffusion property. Computing reputation of another within a network requires knowledge of degrees of its neighbors. We develop an algorithm for estimating degrees of each neighbor. The algorithm considers observations associated with neighbors as a Bernoulli trial and repeatedly estimate degrees of neighbors as a new observation occurs. We experimentally show that the algorithm can compute the degrees of neighbors more accurately than a simple counting of observations. Finally, we design a bayesian reputation game where reputation is used as payoffs. The game theoretic view of reputation computation reflects another level of reality in which all agents are rational in sharing reputation information of others. An interesting behavior of agents within such a game theoretic environment is that cooperation- i.e., sharing true reputation information- emerges without an explicit punishment mechanism nor a direct reward mechanisms.

REPUTATION COMPUTATION IN SOCIAL NETWORKS AND ITS APPLICATIONS

By

JooYoung Lee

B.S. Korea Advanced Institute of Science and Technology, 2007

Dissertation

Submitted in partial fulfillment of the requirements for the
degree of Doctor of Philosophy in Computer Science
in the Graduate School of Syracuse University

August 2014

Acknowledgments

First, I would like to thank my advisor, Professor Jae C. Oh who taught me with patience for last 7 years. He guided me whenever I was lost, and believed in me through all years. I also have to thank my committee members, Dr. Mohan, Dr. Mehrotra, Dr. Moon, Dr. Tang and Dr. Yu for their precious comments and support. Syracuse has been a great place for me and it will be my second home forever. My special thanks go to my dearest friends in Syracuse who helped me endure hard times and gave me endless support. I dedicate this thesis to my family, my Mom and Dad, my sister and brother who are always there for me. I love you all dearly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Reputation management plays an important role in online communities that include e-commerce web sites, such as *e-bay* and *amazon.com*, peer-to-peer computing environments [1], and online social networks [2]. Existing reputation management schemes often require users to explicitly rate each other to compute reputations. For example, the simplest way of computing reputation is to average all the ratings a user receives from other users as in *amazon.com*'s 5-star rating system. However, in general, these rating systems have the following weaknesses: (1) systems cannot force users to rate each other, and (2) consequently, not all user interactions contribute to ratings, resulting inaccurate calculation of reputation. In this thesis, we show that it is possible to compute reputations of users by analyzing their reference behaviors in a social network that is built by extracting key contents of documents. Our method can extract reputation based on users' interactions manifested in the constructed social network.

Reputation management can also be useful in rating documents in their importance. In processing a large amount of unstructured data such as web documents and emails, identifying author's reputation can help in extracting important information. For example, an *automatic document summarization* technique can extract key phrases and return a shortened vesion of the original text. Before automatically summarizing documents, one could

filter out less important documents with the additional aid of *author reputation*. Visualization is another example of presenting textual data in a schematically abstracted graphical form [3]. When visualizing a network of relationships among texts in a graphical form, one could associate quantitative measures of reputation with nodes and edges of the graph.

Accurate reputation information about nodes in social networks improves services provided by the networks. For example, reputations can be calculated and updated for web sites and servers to identify malicious nodes and connections. The ability to observe and analyze propagations of reputations within a large social network structures is also important.

All our presented algorithms in this thesis take a micro-level approach and we use the terms distributed approach and node-centric approach interchangeably.

In Chapter 2, we present some of the existing reputation computation models of our interest.

In Chapter 3, we develop two automatic reputation computation schemes using knowledge extraction from unstructured emails through constructing a social network of authors. The first algorithm computes reputation only considering direct reference behaviors of authors. The second algorithm goes one step further and incorporates indirect references in the computation. The social network based algorithms are also tested on classification of emails.

In Chapter 4, we present a new reputation management model that is suitable to represent the emergence and propagation of reputations within social network structures. The algorithm considers *frequencies* and *velocities* of interactions online. Also, through the experiments, we show that how reputations emerge and propagate in random social networks and how the model captures the idea of dynamic reputation propagations from one part of the network to another. We also show experiments on real *Autonomous Systems Networks (ASN)* of the Internet for identifying malicious ASN nodes through our model. We compare our results with an existing reputation values computed by another well accepted ASN

reputation management system. The results show that our algorithm computes similar reputation values as the values provided by the existing algorithm. However, we show that our algorithm is better suited to find many malicious ASN nodes while the compared method is good for finding the worst malicious node only.

In Chapter 5, we develop a distributed algorithm that estimates degrees of neighbors as degrees of nodes in social networks are important information when computing reputations of users. Since the degree information of nodes is not public in most cases, we present the degree estimation algorithm based on Beta distributions.

In Chapter 6, we formulate a bayesian game where reputation is part of payoffs and the types of players are decided by their reputation and degree values. Through this game, we can predict the outcomes of interactions among players when reputation and degree values are applied. We show that cooperation among players emerges as the game is repeated.

1.1 Contributions

We present main contributions of this thesis.

- We develop algorithms to automatically compute author reputation from unstructured data.
- We develop a distributed reputation computation model, *ReMSA*.
- We design a node-centric algorithm to estimate degrees of neighboring nodes.
- We design a bayesian reputation game that models cooperative behaviors of selfish players.

CHAPTER 2
REPUTATION COMPUTATION MODELS

Online communities, such as social networks, electronic markets and distributed peer-to-peer systems, bring together people who may not know each other in the physical world. Since they have limited information about each other, there is a need for quantifications of trustworthiness. To overcome this uncertainty, reputation mechanisms are widely adopted for trust management in online environments. Reputation management systems are employed in many different applications to help predict the future behaviors of online entities based on their past activities.

The most general structure for computing reputation is combining the direct experiences with a subject and information received from others. Often the direct experiences are weighted by a time decaying factor and the third-party information is weighted by the credibility of the source. Evaluating the direct experiences is the essential and application specific part of reputation computations. Different reputation systems have various ways of evaluating interactions between agents and combining all the components to produce reputation values. In this Chapter, we study how some of popular reputation systems compute reputations.

In computing reputations, there are centralized, also referred to as macro-level, and distributed, also referred to as micro-level, approaches. Centralized reputation computa-

(a) (b)

Fig. 2.1: Classification of reputation computation models.

tion models are often used in commercial applications, such as *eBay* and *Amazon*, where centralized authorities are explicit, but such approaches fail in environments that lack a central authority. In networks such as the Internet, social networks and other multi-agent environments, a distributed reputation algorithm is naturally more suitable. Previously, several distributed reputation algorithms including *AFRAS* [4] and *HISTOS* [5] have been presented. However, these two algorithms do not consider frequencies and *velocity* of interactions. Velocity of interactions measures the second order information of frequency, i.e., the rate of changes in frequency. These algorithms also lack the consideration of topological information of the networks and the subjectivity of reputations. (i.e., two different nodes may perceive the reputation of a node differently.)

2.1 SPORAS and HISTOS

SPORAS is a reputation mechanism for loosely connected communities and *HISTOS* is a reputation mechanism for highly connected communities, introduced by Zacharia [5]. *SPORAS* computes a global reputation values for each user in the community based on ratings from users. Unlike *SPORAS*, *HISTOS* computes reputations of users based on who makes the query, and how the person rated other users. Therefore, a path has to exist from the person who requested reputation value and the other person whose reputation value is

in question. If there doesn't exist a path, it falls back to *SPORAS*. Each user starts with a minimum reputation in *SPORAS* and therefore the reputation value of a user never falls below the reputation of a new user. In our algorithm, introduced in Chapter 4, reputation values of new users start from the neutral value, zero, where reputation values range from -1 to 1. This initial setting of starting from the minimum reputation value could influence new users behaviors as it gives an incentive to create ill intentions.

HISTOS computes personalized reputation values depending on who makes the query and how the person rated neighbors. For example, if a person A wants to know B's reputation value, first, A finds all directed paths connecting A to B. Then A recursively aggregates the reputation value of the person multiplied by the rating he gave to the target. This process can only be done when A knows all the ratings of the users connecting to B in the path. Since *HISTOS* is designed to be applied to electronic commerce systems, it assumes the ratings and reputation values of users are visible to everyone. It also assumes that the network structure is known to public since one has to find the shortest paths to compute the reputation value of the target. Our algorithm aims to compute completely distributed reputation values and does not make any of the assumptions made in *SPORAS* and *HISTOS*.

2.2 AFRAS

AFRAS is a multi agent system devoted to manage reputation using fuzzy logic [4]. The fuzzy value consists of four squares that define a trapezium. Each square represent human-like attributes which are susceptibility, sociability, shyness and remembrance. The remembrance value is updated after each interaction according to the success of the last prediction of reputation. In our case, remembrance is represented with a time decaying factor which balances the weights of the previous reputation value and the current feedbacks. We also consider, in our algorithm, the rate of interactions to emphasize frequently occurring interactions.

2.3 FR TRUST

FR TRUST is a fuzzy reputation computation model for semantic P2P Grids [6]. The model is specifically targeted for semantic P2P environments where nodes are clustered based on their semantic similarities. On a P2P grid system, there are virtual organizations to which nodes belong and each virtual organization is represented by a special node called group coordinator. After nodes in a virtual organization have some interactions with a peer in question, they report their own evaluations about the peer to the group coordinator which decides if the peer is malicious based on a threshold. The model is centralized in a way that there is some authorities who collect reputation scores, but it is also distributed in a sense that each node act as judges for the peer and report individual scores to the super node. There is also a special agent called trust agent who is responsible for storing reputation scores and computing global reputation value for nodes. However, this model can only be utilized in a confined architecture where group coordinators and trust agents exist. Our approach aims more general environments with no explicit topological restrictions.

2.4 TAUCA

TAUCA is an anomaly detection scheme for feedback-based reputation systems [7]. *TAUCA* is not a reputation computation model itself but it helps reputation systems to identify malicious users who try to manipulate the systems by submitting false reviews and recover reputation scores. *TAUCA* first uses a change detector to detect suspicious time intervals in which attack may be present. Then, the suspicious group of users are identified by calculating Euclidean distance of ratings given by each pair of users. Finally, *TAUCA* removes ratings from malicious users from the system. We assumed ratings given by users in the system are truthful in Chapter 4, but we relax the assumption and discuss the behaviors of users when there are dishonest users in Chapter 6. We show that truthful cooperations are sustained without explicit detection schemes if all users are rational and future concerned.

2.5 Terminologies: ratings, evaluation, and opinions

In Chapter 3 and 4, we introduce a macro-level and a micro-level reputation computation models and their applications, respectively. Usually, in macro-level approaches, **ratings** are the values given explicitly to each other or to systems which are used to compute reputations of users. However, in micro-level approaches, there is no explicit **ratings** given to assess the quality of interactions. Instead, we use a term **evaluation** since the assessment value is not shared or given to other users. To supplement subjective, therefore biased, nature of **evaluation**, we introduce a recursive voting mechanism in Chapter 4 to collect **opinions** of other users when reputation values are computed. *Opinions* are the computed reputation values from other users that are explicitly given to a user per request. In summary, in macro-level models, **ratings** are aggregated to compute **reputation** of a user, while in micro-level models users compute **reputation** as a weighted sum of **evaluation** and **opinions** from other users. Therefore, **reputation** becomes a global knowledge in macro-level approaches whereas **reputation** values are private values of users who computed them in micro-level approaches.

CHAPTER 3
AUTOMATIC REPUTATION COMPUTATION THROUGH DOCUMENT ANALYSIS

In this Chapter, we introduce two social network based algorithms and automatically compute *author reputation* from a collection of textual documents. To compute reputations of authors without explicit ratings from each other, we extract *keyword reference behaviors* of the authors to construct a social network, which represents relationships among the authors in terms of *information reference behavior*. With this network, we apply the two algorithms: the first computes each author's reputation value considering only *direct reference* and the second utilizes *indirect reference* recursively. We compare the reputation values computed by the two algorithms and reputation ratings given by a human domain expert. We further evaluate the algorithms in email categorization tasks by comparing them with machine learning techniques. Finally, we analyze the social network through a community detection algorithm and other analysis techniques. We observed several interesting phenomena including the network being scale-free and having a negative assortativity [8].

3.1 Document classification

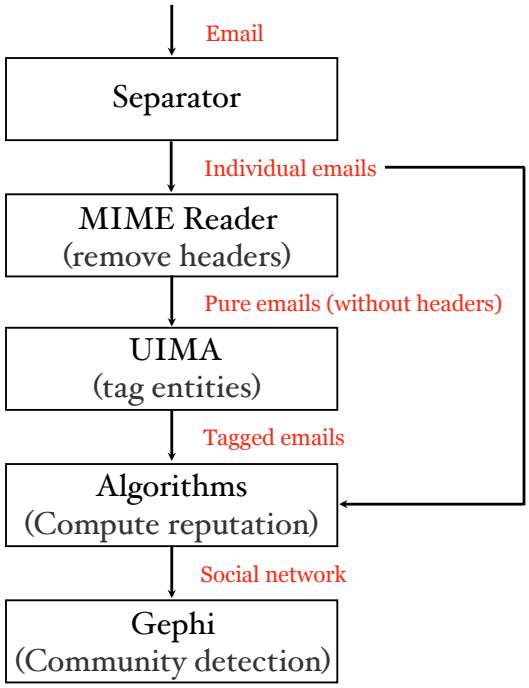


Fig. 3.1: Original data is one long text file containing many emails. We separate emails using a separator program to individual files. We remove unwanted information from individual emails using simple MIME Reader to get Non-MIME formatted emails. These emails are further processed by UIMA to get tagged emails for the social network algorithms. We build a social network of authors based on their reference behavior and use Gephi to visualize the network information.

We utilize a collection of emails to construct social network and compute reputation. The emails are from three mailing lists which specialize in security issues and collected for over a month period with a total of $2,415$ individual emails. In order to construct a network to compute reputation of authors, we preprocess the emails. Figure 3.1 shows the main steps of preprocessing. First, we start with a very long text file containing a series of raw emails. Then we separate the file into individual emails, one file per email, using our separator program, written in Java. These individual emails are one of the two inputs to the reputation computation algorithm. Next, we remove the header and signature information from emails using a MIME (Multipurpose Internet Mail Extensions) reader to

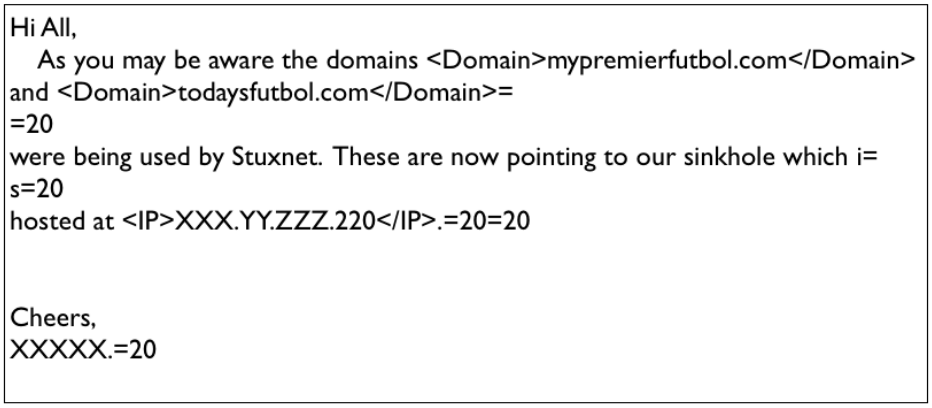


Fig. 3.2: An example of a tagged email using UIMA; this figure shows two DOMAIN tags and an IP tag.

eliminate noise. We then use UIMA (Unstructured Information Management Architecture) to tag entities such as IP, URL and DOMAIN in the emails. Figure 3.2 shows one of the tagged email by UIMA. [1] UIMA is an open source architecture that analyses unstructured documents, video and audio. We wrote a UIMA descriptor that identifies IP, URL and DOMAIN as well as email addresses of authors. We then convert the UIMA annotated outputs into tagged emails. We construct a social network of authors using both individual tagged emails and emails with headers. The purpose of emails with headers is to extract dates and time of the emails to extract reference behaviors among authors.

3.2 Constructing Social Networks and Computing Reputations

We introduce how we build a social network of authors from the email data. We also discuss two reputation computation algorithms.

[1] Details of the email are deleted for confidentiality.

3.2.1 Building a network

An author reputation social network is a weighted digraph where vertices represent authors and weighted edges represent reference behaviors. The weighted edges are computed as described in Section 3.2.2 and 3.2.3. Such a network is built from a time-stamped collection of *documents*. Next, in order to present the network building algorithms we define several convenient functions. Following these definitions we give an example of applying the functions to a set of documents (emails).

In the application at hand, where we build a network from a collection of email text files, a *document* is an email, where we assume for each document d that a single *author*, denoted by $\mathsf{author}(d)$, and a single time stamp, which we represent by a real number and denote by $\mathsf{time}(d)$, is extractible from each email. Define a function $\mathsf{authors}$ that maps sets of documents to sets of authors by

$$\mathsf{authors}(D) = \bigcup_{d \in D} \{\mathsf{author}(d)\}.$$

Also, where D is a set of documents, let

$$\mathsf{times}(D) = \bigcup_{d \in D} \{\mathsf{time}(d)\}.$$

Assume that each document contains one or more *entities*, the nature of which may be left as a parameter to be instantiated later. We also assume that the set of entities, $\mathsf{entity}(d)$ contained in a document d is extractable from d. Again, where D is a set of documents, let

$$\mathsf{entities}(D) = \bigcup_{d \in D} \mathsf{entity}(d).$$

Note that in the above definitions, $\mathsf{author}(d)$ and $\mathsf{time}(d)$ are not sets, whereas $\mathsf{entity}(d)$ is a set.

While a document d uniquely determines a time t and an author a, the converse deter-

mination, a time/author pair *(t,a)* uniquely determines a document, is also true, assuming an author can generate only one document at a time (although the time stamps associated with any particular author may be in rapid succession.) Therefore, a document is a partial function of a time/author pair. (The function is partial because an author *a* may not have generated a document at a particular time *t*). We will, in the sequel, regard time/author pairs *(t, a)* as documents. Think of *(t, a)* as the *undefined document*, if there is no document in the set *D* of documents input to our algorithms with both time stamp *t* and author *a*.

Figure 3.3 shows three email headers extracted from the real separated email documents:

email header1
Email Subject: additional information from First Citizens
Email creation time: Wed Mar 09 18:20:18 GMT 2011
Sender email:owner@xxx.net

email header 2
Email Subject: Money Mule and cards
Email creation time: Tue Mar 08 20:34:22 GMT 2011
Sender email:owner@yyy.net

email header 3
Email Subject: Mule account!
Email creation time: Fri Mar 04 23:14:00 GMT 2011
Sender email:owner@xxx.net

Fig. 3.3: Extracted email headers.

From the previous email headers, we can extract:

$$\mathsf{authors}(D) = \{\text{owner@xxx.net}\} \bigcup \{\text{owner@yyy.net}\} \bigcup \{\text{owner@xxx.net}\}$$

$$= \{\text{owner@yyy.net, owner@xxx.net}\}$$

$$\mathsf{times}(D) = \{\text{"Wed Mar 09 18:20:18 GMT 2011"}, \}$$

"Tue Mar 08 20:34:22 GMT 2011",

"Fri Mar 04 23:14:00 GMT 2011"}

Therefore, we get six pairs of *(t, a)*, three (in *italic*) of which correspond to undefined documents:

(Wed Mar 09 18:20:18 GMT 2011, owner@xxx.net),

(Fri Mar 04 23:14:00 GMT 2011, owner@xxx.net),

(Tue Mar 08 20:34:22 GMT 2011, owner@yyy.net),

(Tue Mar 08 20:34:22 GMT 2011, owner@xxx.net),

(Wed Mar 09 18:20:18 GMT 2011, owner@yyy.net),

(Fri Mar 04 23:14:00 GMT 2011, owner@yyy.net).

A document uniquely determines an author and time but the reverse doesn't hold. We consider authors as nodes when constructing the network later on, so the reverse need not hold (we don't need to know which documents entities come from as long as they belong to the same author).

Algorithm 1 takes as input a finite sequence of 4-tuples, each of which is a *well-formed Information entity*. An *Information entity* is a 4-tuple *(e, t, a, b)*, where *e* is an entity, *t* is a time, *a* is an author and *b* is a boolean. *(e, t, a, b)* is *well-formed* iff *(t, a)* is a defined document, $e \in \mathsf{entity}(d)$ and $b = \mathsf{initial}(t, a)$ where

$$\mathsf{initial}(t, a) = \begin{cases} \textit{false}, & \text{if } (t, a) \text{ is a reply/forward} \\ \textit{true}, & \text{otherwise} \end{cases}$$

Pseudo code for the network building algorithm is given in Algorithm 1. Again, the input to the algorithm is a finite sequence of information entities, *I*, and the output is the social network (Once entities are extracted, we need not know which documents they are

coming from since *information entities* have all the information we need.).

Algorithm 1 Build Social Network(*I*)

for each element (e_i, t_i, a_i, b_i) in I **do**
  $a \leftarrow$ the first author who mentioned e_i
  if a = a_i **then**
    {self–referencing, next element in I}
    break
  end if
  if b_i =FALSE **then**
    {e_i is in a reply or forward document}
    make a connection from a_i to a with weight 1
  end if
  if b_i =TRUE **then**
    {e_i is in an original document}
    make a bidirectional connection between a_i and a with weight 2
  end if
end for

An example of an instance of a sequence of three *Information entities* is given below. The example extracts entities from the email shown in Figure 3.2. The email has 3 entities; *mypremierfutbol.com*, *todaysfutbol.com* and *XXX.YY.ZZZ.220*. The timestamps, author and boolean value for all three are the same since they belong to the same email.

(mypremierfutbol.com, 22 Jul 2010 13:52:08, s@X.com, FALSE)

(todaysfutbol.com, 22 Jul 2010 13:52:08, s@X.com, FALSE)

(XXX.YY.ZZZ.220, 22 Jul 2010 13:52:08, s@X.com, FALSE)

When multiple authors have a common entity in any of their emails, a directed edge exists *to* the source author, whose email precedes others in terms of the time sent, *from* a destination author whose email has the same entity with the source email. More specifically, as shown in Figure 3.4, when more than one author has a common entity, e_1, and if it is the case where author a_1 mentioned the entity e_1 and author a_2 also mentioned e_1 as a reply or forward, a_1 gets an incoming edge from a_2 with weight 1. Otherwise, if e_1 was mentioned by another author a_3 not as a reply or forward, a_1 will get another incoming edge from a_3 with weight 2 and a_3 will also get an incoming edge from a_1 with weight 2. Au-

thors mentioning common entities supports the fact that those entities are hot issues and the author who brought the issue first gets the credit. We only give a positive weight to directed edges because regardless of the context, either positive or negative, authors mentioning a common entity increases the popularity of the entity. The rationale for independent reference getting twice the weight is that since all the authors involved in independent reference are originals, their importance is identified as originators, unlike the authors of replies and forwarded messages. These weights are the basis for computing reputations of authors in algorithms described in 3.2.2 and 3.2.3.

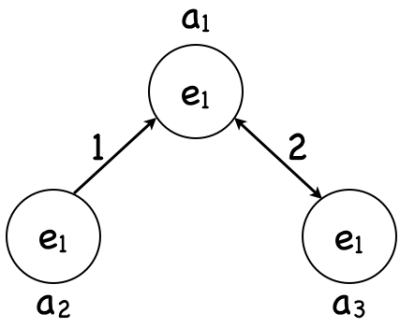


Fig. 3.4: A social network of three authors; a_1, a_2, and a_3. All three of the authors mention the same entity, e_1. a_2 references a_1 as a reply or forward while a_1 and a_3 reference each other independently.

We have developed two algorithms to calculate the reputation of each author; one uses only direct references and the other uses indirect references as well. Both algorithms run on the network built from the previous algorithm.

3.2.2 A Sporas-based algorithm (Direct reference)

The first algorithm we propose is based on *Sporas*, a reputation mechanism for loosely connected online communities [5]. *Sporas* updates user's reputation (R) upon each rating given by another user. Ratings given by users with high reputation are weighted more. Since our application does not assume a centralized environment where the system can ask users to rate each other whenever they have interactions, we adopted reference behaviors

as a way of giving ratings to other users. Therefore, from the social network we built, a node, which represents an author, has a reputation based on incoming edges it gets.

The reputation value for each author is computed as follows:

$$R_{t+1} = \frac{1}{\theta} \sum_{i=1}^{t} \Phi(R_i) \times R_{i+1}^{other} \times \frac{W_{i+1}}{2}$$

$$\Phi(R) = 1 - \frac{1}{1 + e^{\frac{-(R-D)}{\sigma}}}$$

where,

t is the number of references the author has received so far,

θ is a constant integer greater than 1,

W_i represents the rating given by the user at time *i*,

R^{other} is reputation of the author who is referencing *R*,

D is the range of the reputation values,

σ is the acceleration factor of the damping function Φ.

The smaller the value of σ, the steeper the damping factor $\Phi(R)$.

For experiments, we used θ = 3 and σ = 0.5. The maximum value for the reputation is 5 and the default value is 1. The damping function $\Phi(R)$, ensures that the reputations of trustworthy persons are more robust against temporary malicious attacks. The value of θ determines how fast the reputation value of the user changes after each rating. The larger the value of θ, the longer the memory of the system. For detailed information on *Sporas* algorithm, refer to [5].

3.2.3 The Indirect Referencing algorithm

The second algorithm is based on the *TrustMail* rating system [9]. *TrustMail* calculates the reputations of incoming emails based on human ratings. Since the original algorithm was not designed for distributed settings, authors do not have representative reputation values. In the TrustMail system, reputation is computed only when a user asks for another's reputa-

tion value. The reputation values depend on the relationship between all of the requester's neighbors and the destination node (i.e., the node being evaluated). Consequently, reputations are relative. Since we want authors to have objective reputation values–so that we can have results to compare with the first algorithm–we evaluate all the relative reputation values for each author. In other words, we run the indirect algorithm for each author node as if each node is asking for everyone else's reputation values. Algorithm 2 describes how reputation can be inferred when the *source* is asking for *sink*'s reputation value. We then average out the reputation values since we accumulate all the reputation values from all the neighbors a node has.

Algorithm 2 getRating($source, sink$)

```
mark source as seen
if source has no rating for sink then
  denom = 0, num = 0
  for each j in neighbors(source) do
    if j has not been seen then
      denom++
      j2sink = min(rating(source, j), getRating(j, sink))
      num += rating(source, j) * j2sink
      mark j unseen
    end if
    rating(source, sink) = num/demon
  end for
  return rating(source, sink)
end if
```

The main idea that given source *i* and sink node *s*, if *i* has direct edge to *s* then no inference is necessary. If there is no direct edge between *i* and *s*, *i* forwards the query to all the neighbors, namely *j*. The algorithm calculates t_{is}, the relative reputation of the *sink* for the source *i*. The condition in this formula ensures that the source will never trust the sink more than any intermediate node.

$$t_{is} = \frac{1}{n} \sum_{j=0}^{n} \begin{cases} (t_{js} \times t_{ij}), & \text{if } t_{ij} \geq t_{js} \\ t_{ij}^2, & \text{if } t_{ij} < t_{js} \end{cases}$$

3.2.4 Applying time decaying function to the algorithms

There could be cases where multiple authors independently discuss the same entity. As discussed in Section 3.2.1, the dependency relationship, the direction of an edge, is determined by the timestamp. Whoever has mentioned the entity earliest gets credit for the originality, whereas in the independency relationship, all the involved authors get credit regardless of time in independency relationship. Consider the case in Figure 3.5. In addition to a_1 and a_3, a_4 mentions the same entity, e_1, say, a week later. According to the *Sporas-based algorithm*, both a_4 and a_1 should get incoming edges with weight 2. However, if the latter independent reference, which is a_4 mentioning e_1, happens after a sufficient amount of time, it is reasonable to consider the latter reference as a new topic rather than relating it to the previous reference.

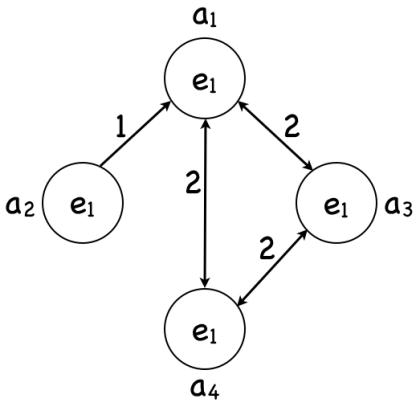


Fig. 3.5: Independency relationship; a_1, a_3 and a_4 get bidirectional edges from each other since they all contain e_1, independently.

To accommodate this issue, we incorporated a *time decaying function* shown in Figure 3.6. When a new entity is introduced by a source author and shortly referenced by others, there is a high chance that the references are related, but the relevance decreases over time. Therefore, after a sufficient amount of time has passed, we consider the entity to be independent from previous references. We use the *cosine* function to capture this idea. According to our *time decaying function*, when a new entity, e_2, is mentioned by the first author and independently referenced by another author immediately, they will both get

incoming edges with weight 2 (technically, the latter author will get an edge with weight slightly less than 2 by the function). As time passes, authors who independently references e_2 will have incoming edges with weight less than 2 according to the *time decaying function*, at worst case with weight 1, when the topic has completely died out. Now, the edge weight of an independent reference is calculated as follows.

$$weight = 0.5 \times \cos(period \times (t_2 - t_1)) + 1.5$$

where, t_1 is the time when original author introduce an entity, e, t_2 is the time when a new author independently references e, *period* is $\frac{604800000}{2} \times \pi$.

604800000 is a week in milliseconds and *period* is set so that the period of the *cosine* function be a week.

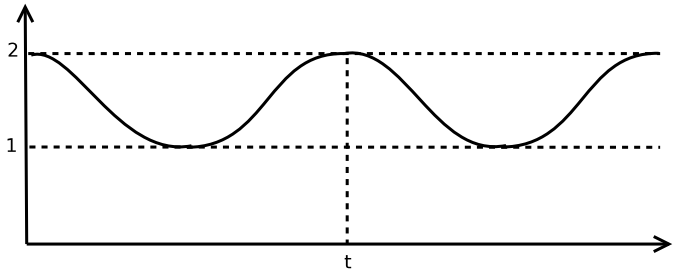


Fig. 3.6: Time decaying function: $cos(period \times t)$

3.3 Experiments: Email Author Network

From 2415 emails, we have extracted 426 authors. In Figure 3.7, we show the reputation values of all the authors, sorted in descending order from the perspective of the second algorithm. For both algorithms, the reputation of authors in the top and bottom tiers tend to agree more than the middle ones.

The comparison between the two algorithms with the *time decaying function* is shown in Figure 3.8. Also Figure 3.9 shows differences in reputation values using the *time decaying*

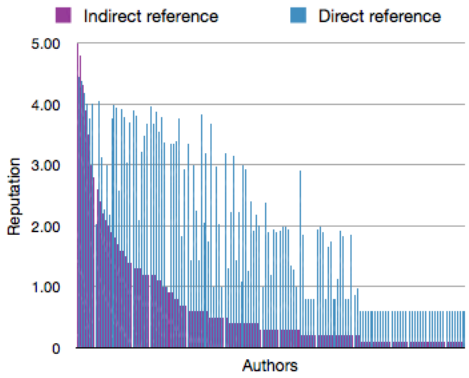


Fig. 3.7: Reputation of authors before applying the *time decaying function*.

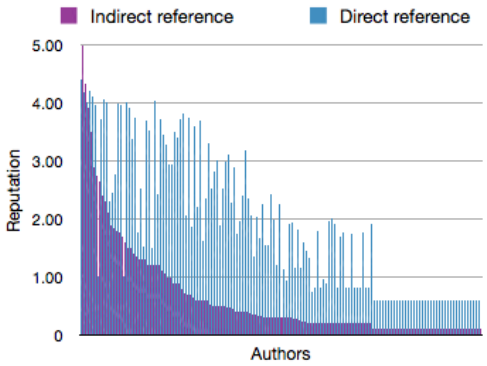


Fig. 3.8: Reputation of authors after applying the *time decaying function*.

function or not using. We only picked 15 authors here, since visualizing all 426 authors would be messy. We picked five authors with high reputation, five authors with middle reputation, and five authors with low reputation. Some differences from authors with high reputation were zero and that's why some values are not shown. For authors with high and low reputations, the effect of *time decaying function* was minimal. Authors with the mid-reputation range show that the difference between two algorithms is smaller in most cases after the *time decaying function* is applied.

We also compared human assigned reputations and reputations computed by the algorithms. Eleven authors were picked and assigned reputations by a human domain expert. Figure 3.10 shows how the reputations given by the domain expert compared to the reputations computed by the algorithms before applying the *time decaying function*. Figure

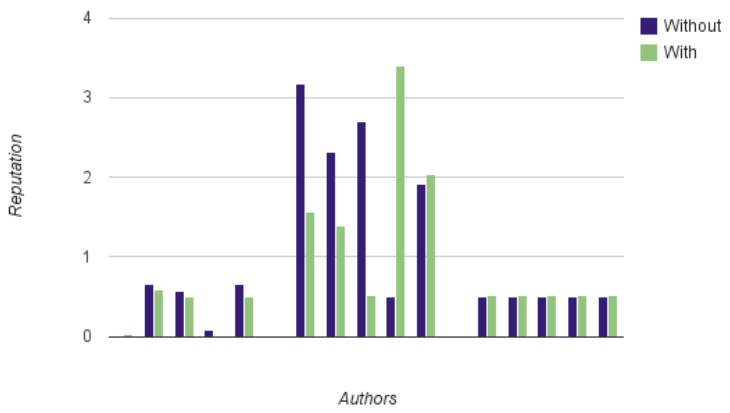


Fig. 3.9: Effect of using time decaying function. We show differences of the reputations given by the two algorithms before and after applying the *time decaying function*. We pick 15 authors to compare; first 5 entries represent authors with high reputations, next 5 entries represent authors with middle reputations, and the last 5 entries represent authors with low reputations.

3.11 shows the result with the *time decaying function*. It is hard to conclude whether one is superior to the other since only eleven authors' reputation values are available from the domain expert. This difficulty motivated us to test our algorithms further; we compare the two algorithms with a decision-tree based machine learning algorithm in categorizing emails in Section 3.3.1.

In Figure 3.12, we ordered 10 authors in increasing order of the ratings given by the domain expert. As seen in Figure 3.12(a) and 3.12(b), reputation values of authors are more comparable with the ratings after applying the time decaying function.

In summary, we have shown that the two algorithms produce agreeing reputation values; but the reputations assigned by the domain expert shows small divergence from the reputations given by the algorithms. Possible explanations include the human expert may have assigned higher reputation values to recognized authors or authors given high reputation by the domain expert were not active in writing important emails during the time the data were collected (one month). For future research, we plan to gather email data ranging over a year and study how experiment results change. It would be interesting if we

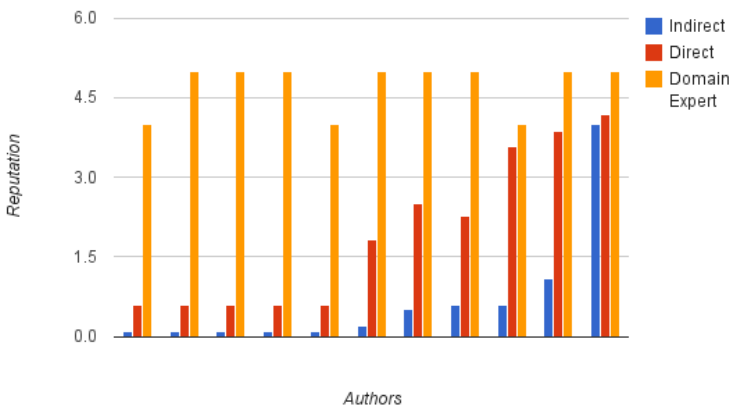


Fig. 3.10: Human assigned reputations compared with the algorithm based reputations without the time decaying function.

divide the period into three so that each has four months of email data and compare how reputations of authors change over time.

A word of clarification may be needed about the tail ends of Figure 3.7 and 3.8. In Figure 3.7 and 3.8, the tail with equal values represents authors with the default value. The difference only exists because author reputation given by the two algorithms were in different ranges before normalization. Since reputations given by the second algorithm go up to 255, even if an author has the same default reputation value from both algorithms, which is 1, when normalized, reputation given by the second algorithm appears to be smaller. The same explanation applies to Figure 3.10 and 3.11 as well as other comparisons. For example, in Figure 3.9, the rightmost short bars from authors with low reputation actually represent no difference between the two algorithms.

3.3.1 Email Categorization Experiments

Generally, reputation results are very hard to evaluate since there is no concrete values to compare with and the values are often subjective. We have used human expert's ratings to compare with outputs from our algorithms but the available number of ratings was not

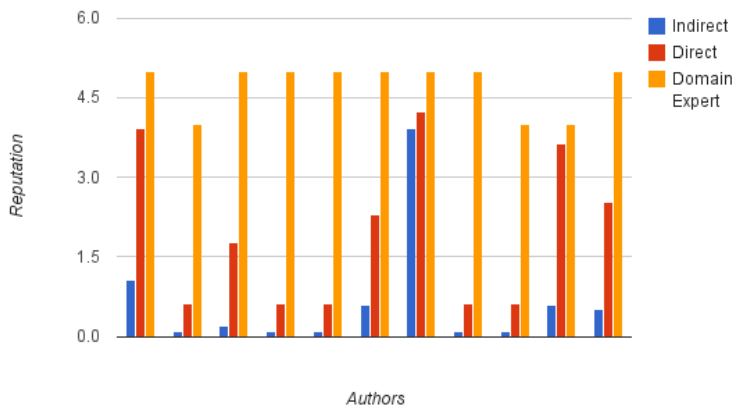


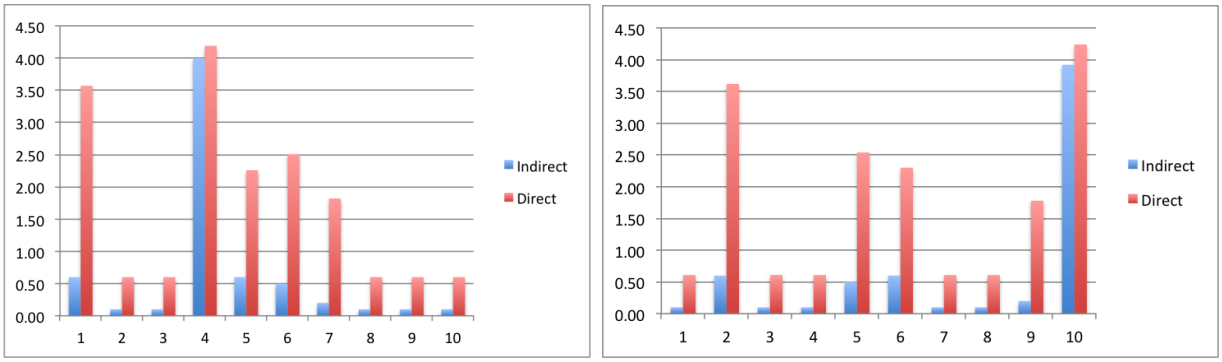
Fig. 3.11: Human assigned reputations compared with the algorithm based reputations with the time decaying function.

enough. To further evaluate the efficacy of our algorithms, we test the algorithms in email categorization task and compare the results with machine learning algorithms.

We categorize emails into three groups: *useful*, *helpful*, and *useless*. We used 16 attributes to categorize emails. The details of each attribute is explained in *Table 1*. We use RapidMiner [10], which is the most widely used open source data mining tool, to train the model. First, we manually categorize 100 samples of emails into the three groups by reading the contents of the emails, without relying on the attributes so that our manual categorization be independent from the machine learning of RapidMiner. Then we train the model with the training set. Figure 3.16 shows the decision-tree model trained.

To build a social network of emails, for each email, we counted the number of entities referenced by other emails. Analogous to the reference behaviors among authors, our intuition is that, if an email has higher *reputation* than others, (i.e., it has been referenced highly) then it is categorized as useful. The reputation of emails ranges from 0 to 10.

Among 709 emails, the decision tree model categorized 537 as *useless*, 67 as *helpful* and 105 as *useful*. As shown is Figure 3.17, most of the emails categorized as *useless* has low *supported score*, i.e., less than 1, and only a few have *supported score* higher than 3. The emails categorized as *helpful* have consistent *supported score* between 2.5 and 3.5.



(a) (b)

Fig. 3.12: Comparison of the order of reputation values computed before and after the time decaying function is applied. 10 authors are ordered by the ratings give by the domain expert.

The *useful* category, as expected, has the highest *supported score* on average, most of these emails having *supported score* of more than 3.5.

3.4 Social Network Analysis

In this section, we analyze the social network constructed in Section 3.2.1. For the purpose of analysis, we used Gephi [11], an interactive visualization and exploration platform for networks and complex systems.

3.4.1 Community Detection

Community detection can reveal interesting facts about social networks. For example, the community structure of a social network can serve as a summary of the entire network, producing an easy to understand visualization of the network. Figure 3.18 is a visualization of our social network. Nodes and edges represent authors and reference behaviors, respectively, in the emails. Colors represent communities. The network has 36 communities and the modularity is between 0.46. A network with modularity of 0.4 or greater has meaningful community structures.

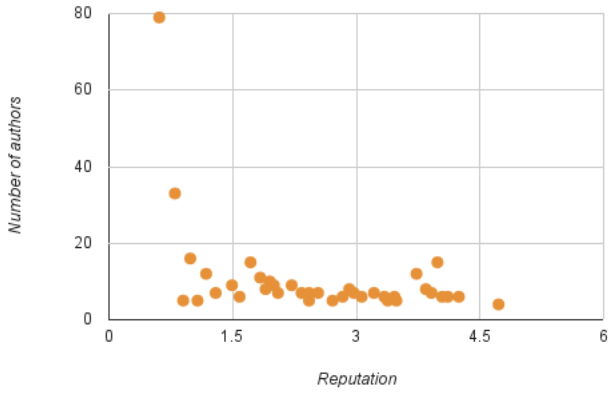


Fig. 3.13: Distribution of the reputations of authors by the direct reference algorithm follows power law distribution

3.4.2 Average Path Length

The network has the *average path length* of 4.1, which is shorter than the "e-mail network"of Ebel et al. [12]. Since email communication doesn't require senders and receivers to closely share certain chracteristics, unlike other networks, such as co-authorship networks, email networks are believed to have lower value of *average path length*. This means that nodes in the network in general are more closely connected.

3.4.3 Scale-free Behavior

A scale-free network is a network with its degree distribution following power law, at least asymptotically. As shown in Figure 3.13 and 3.14, the distribution of reputation in the overall network follows a power-law distribution. Interestingly, each community in the network also follows a power-law distribution as in Figure 3.15. Within each community, there is a "super" author that the members of the community follows. This implies that the network has *negative assortativity* which will be discussed in the next Section.

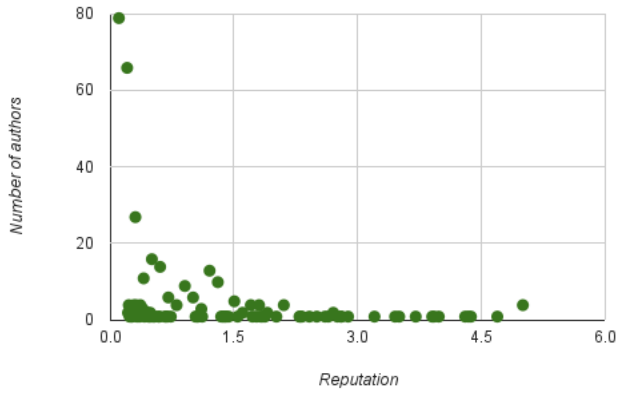


Fig. 3.14: Distribution of the reputations of authors by the indirect reference algorithm follows power law distribution

Testing Power-law Hypothesis.

We use a method described in [13] to test the goodness of fitting the computed reputations of authors to the power-law distribution.

3.4.4 Assortativity

Assortativity is a preference for nodes in a network to attach to others that are similar or different in a metric. We calculated the *assortativity coefficient*, *r*, of the network found. The assortativity coefficient is essentially the *Pearson correlation coefficient* of degree between pairs of linked nodes [14]. The value of *r* is approximately *-0.2* in the network. This is interesting because, unlike many social networks, which have positive assortativity [8], each community in our social network follows a power law distribution. This means that in each community, there is small number of authors with high reputations followed by a greater number of authors with lower reputations as shown in Figure 3.15. This characteristic is shared with citation networks [15]. Another fact is that the network has two obvious clusters as shown in Figure 3.18. The clusters have almost equal number of authors and the degree distributions of nodes for the two clusters are quite similar. The nodes connecting

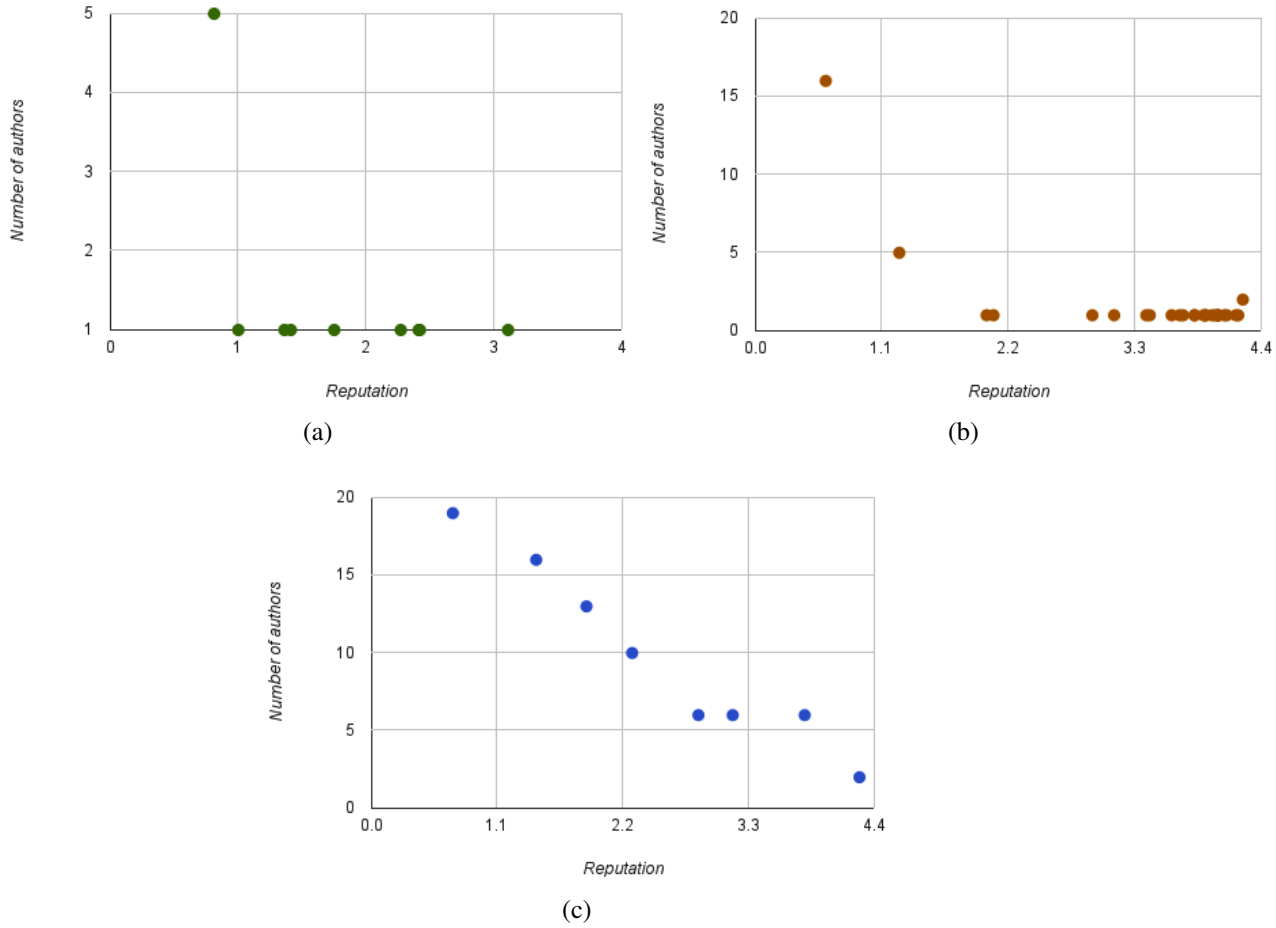


Fig. 3.15: Each community follows power law distribution.

the two cluster may play special roles. We plan to investigate the roles in our future work.

3.5 Chapter Summary

Document processing and social network reputation computation have been around for some years, but combining them to automatically compute reputations of authors and to categorize emails has rarely been done. We developed a method to extract references from contents of documents and to build author reputation network automatically. We also have developed two algorithms for calculating reputations of authors by traversing the network using direct references and indirect references. Our methods can be applied not only to emails but also to other unstructured data such as RSS and proxy logs, which will be our

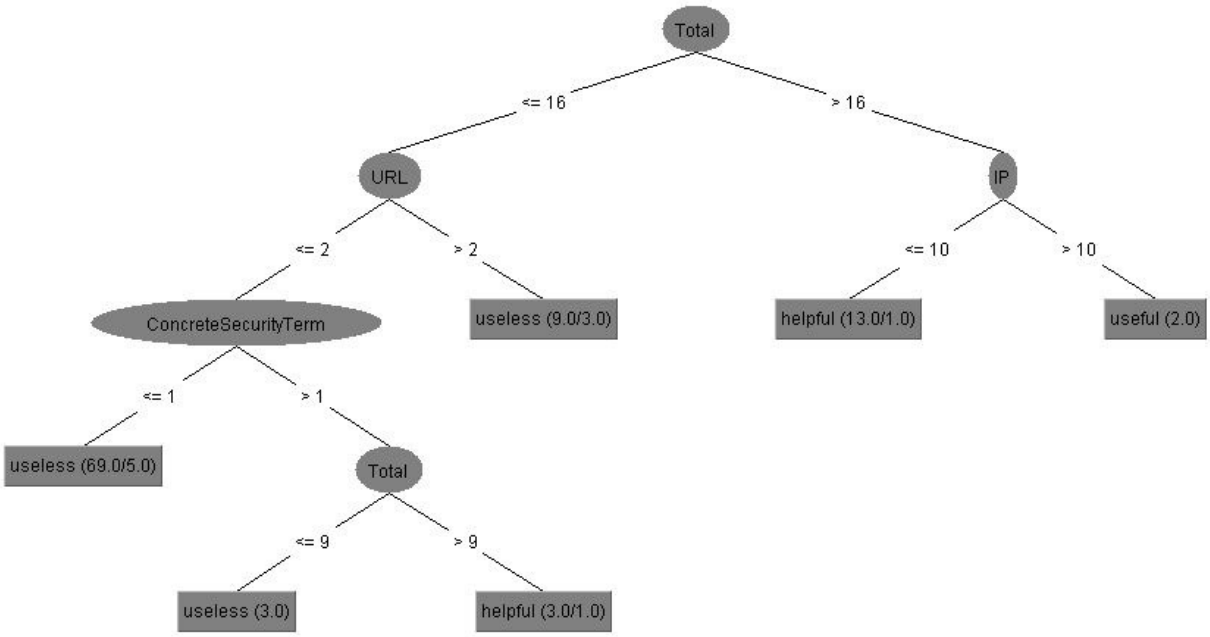
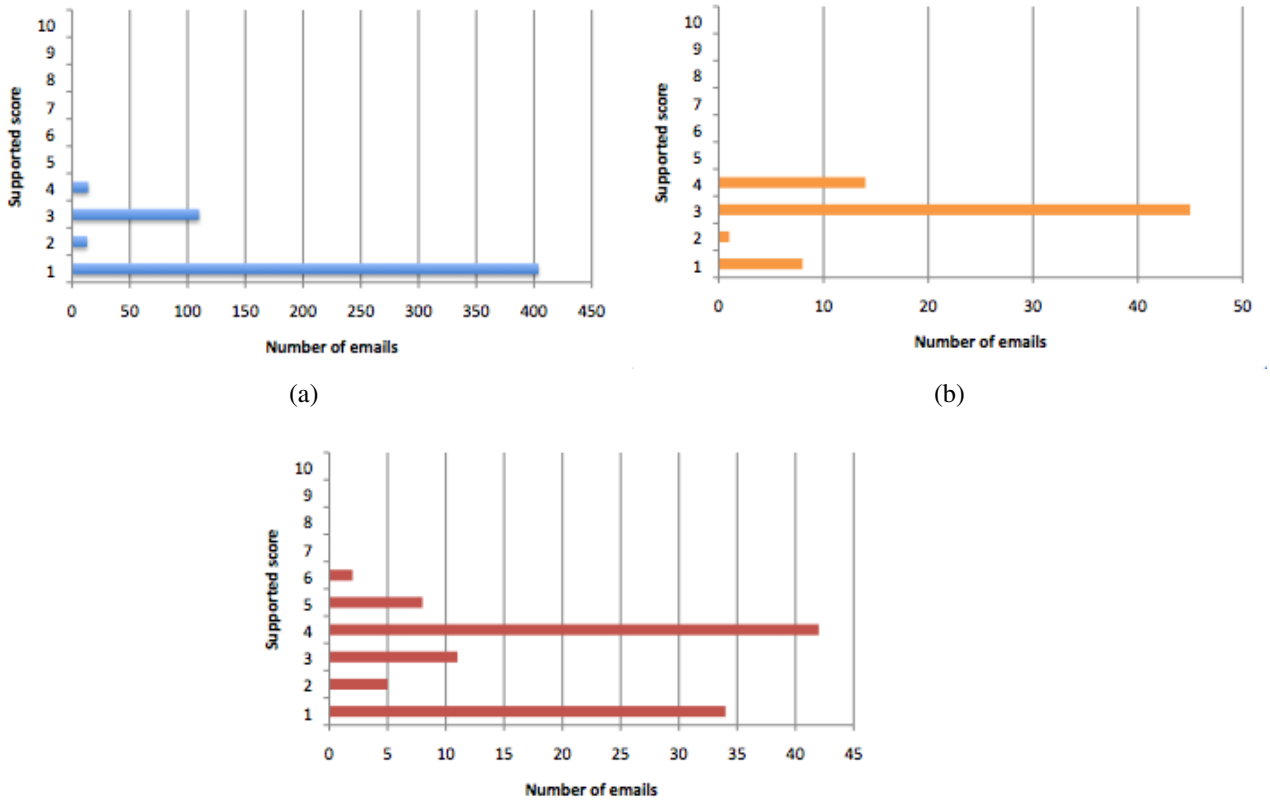


Fig. 3.16: Trained tree model for categorization by Rapidminer.

next step. We analyzed the reputation network generated using community detection. Some interesting properties are identified–such as power-law distribution of author reputations within each community as well as in the global network . In future research, we plan to evaluate importance of documents using automatically extracted author reputation and construct visualization tools that highlight the importance.

Attribute Name	Usage
ConcreteSecurityTerm	Concrete security terms in the email, such as root kit, Zeus
GenericSecurityTerm	Generic security terms in the email, such as threat, Malware
SpecialWords	Special interesting words, such as Russia, Iran
SecurityVerb	Security related verbs in the email, such as attack, hide
Length	Length of each email
RegistrantInfo	True, if the email contains system-generated registrant information
Request	True, if the email is requesting specific information
ReplyToRequest	True, if the email is a reply to any request
Attachment	True, if the emails contains attachment
List	True, if the email contains non-Natural Language formats such as a list
IP	IPs in the email
DOMAIN	DOMAINs in the email
URL	URLs in the email
EMAIL	EMAIL addresses in the email
WinRegistry	True, if the emails contains window registry information
Total	Sum of the number of attributes values (except the attributes that return boolean values)

Table 3.1: Sixteen attributes used for training



(a) (b)

(c)

Fig. 3.17: Comparing categorization result from the machine learning algorithm versus supported score of emails

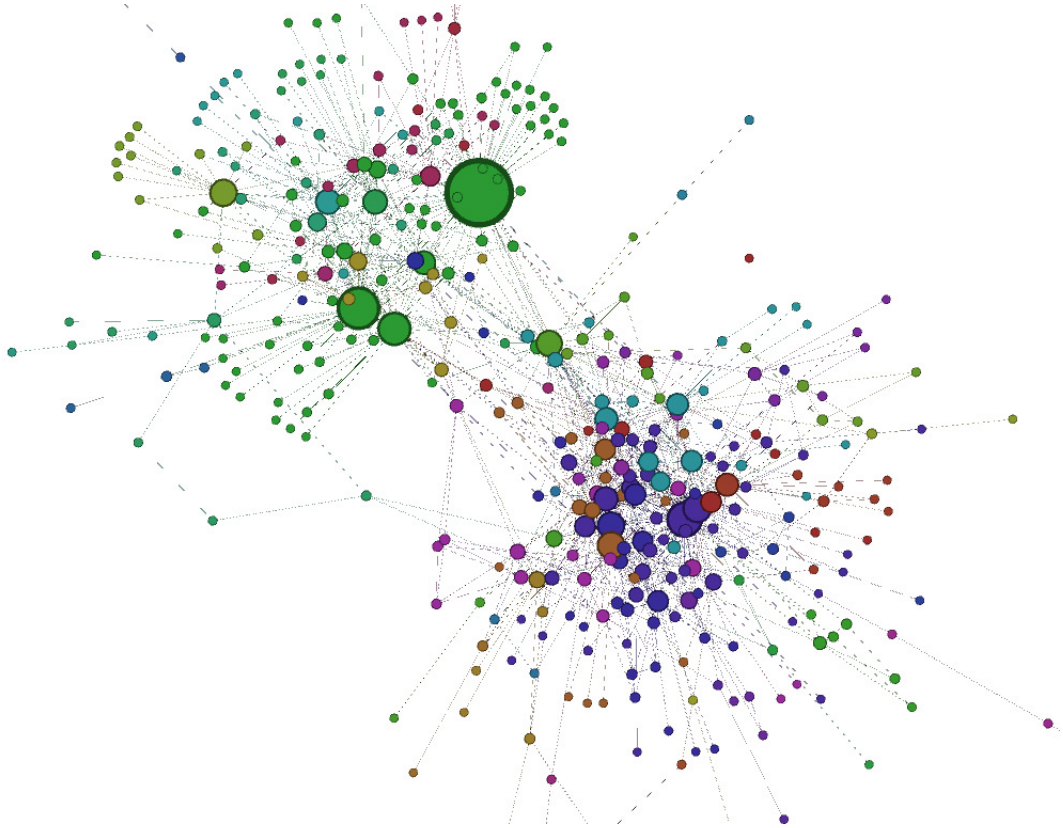


Fig. 3.18: Social network of the authors; nodes are weighted with degrees, colors are partitioned by modularity classes.

CHAPTER 4
A MODEL FOR RECURSIVE PROPAGATIONS OF REPUTATIONS

In online social networks, reputations of users (nodes) are emerged and propagated through interactions among the users. These interactions include intrinsic and extrinsic consensus (voting) among neighboring users influenced by the network topology. We introduce an algorithm that considers the degree information of nodes (users) to model how reputations spread within the network. In our algorithm, each node updates reputations about its neighbors by considering the history of interactions and the frequency of the interactions in recent history. The algorithm also captures the phenomena of accuracy of reputations deteriorating over time if interactions have not occurred recently. We present the following two contributions through experiments: (1) We show that an agent's reputation value is influenced by the position of the node in the network and the neighboring topology; and (2) We also show that our algorithm can compute more accurate reputations than existing algorithms especially when the topological information matters. The experiments are conducted in random social networks and Autonomous Systems Network of the Internet. In addition, we show the efficacies of each component in our algorithm and present their effects on the algorithm.

4.1 Reputation Computation on Social Networks

Accurate reputation information about nodes in social networks improves services provided by the networks. For example, reputations can be calculated and updated for web sites and servers to identify malicious nodes and connections. The ability to observe and analyze propagations of reputations within a large social network structures is also important.

Several distributed reputation algorithms including *AFRAS* [4], *REGRET* [16] and *HISTOS* [5] exist. However, these algorithms do not consider frequencies and *velocity* of interactions; frequency of interactions is an important measure of reputation of the users involved. Velocity of interactions measures the second order information of frequency, i.e., the rate of changes in frequency. Existing algorithms also lack the consideration of topological information of the networks and the subjectivity of reputations (i.e., two different nodes may perceive the reputation of a node differently).

We presents a new reputation management model that addresses the above issues. Our algorithm considers frequencies and velocity of interactions online. Because our algorithm is developed by modeling the behavior of social networks, we show the algorithm can be used as an analytical tool for studying social network behaviors as well as a query tool for retrieving reputation values for specific nodes at a given time. Through experiments, we show that how reputations emerge and propagate in random social networks and how the model captures the idea of dynamic reputation propagations from one part of the network to another. We also show experiments on real *Autonomous Systems Networks (ASN)* of the Internet for identifying malicious ASN nodes through our model. We compare our results with an existing reputation values computed by another well accepted ASN reputation management system. The results show that our algorithm computes similar reputation values as the values provided by the existing algorithm. However, we show that our algorithm is better suited to find many malicious ASN nodes while the compared method is good for finding the worst malicious node only. Finally, we extend the previous work presented in [17] to test the effectiveness of each components in computing reputation values.

4.2 ReMSA: Reputation Management for Social Agents.

In this Section, we present the proposed algorithm – *ReMSA: Reputation Management for Social Agents* [17]. A reputation value is computed when an interaction between two agents occurs. In an interaction, an agent can be an observer, observee, or both. After an interaction, the observer evaluates the interaction to compute the reputation of the observee. If both are observers, they will compute reputations of each other. The more interactions occur between two agents, the more accurate the reputations are computed for the agents. As interactions occur within the network over time, reputations of agents in one part of the network will propagate to another part of the network, much similar to what is happening in real-world social networks. At any given time, any agent can query about reputation of an arbitrary agent. Note that the returned value to the agent may be different from the result of the query initiated by another agent. When there's enough interactions among overall agents in the network, reputations will propagate to the entire network, and more homogeneous views of reputations of agent will emerge.

Figure 4.1 shows the flowchart of reputation computation for an observer node when an event occurs. Following subsections explain each process in Figure 4.1.

4.2.1 Events

We define an event as an interaction between two agents with time information. There are two types of events in terms of who owns the event. When an interaction happens between two agents, if both agents can observe each other's behavior, then both own the event. If only one of the agents can observe the other's behavior, only the observer is the owner of the event. All the following computations are based on the observer agent's point of view.

- The set of agents and events are defined as follows.

$$A = \{a_1, a_2, ..., a_n\}$$

$$E_i = \{e_1, e_2, ..., e_m\}$$

$$e_j = (t_j, a_j)$$

where, a_i is an observer agent, E_i is a set of events that a_i as an observer, and e_j is an event which consists of its time, t_j, and the observee agent, a_j.

- Given a network, we define α and β where nodes are agents and edges are relationships.

$$\alpha_i = \frac{d_i}{\max_{m \in A}\{d_m\}}$$

$$\beta_{il} = \frac{d_l}{\sum_{k \in N_i} d_k}$$

where, N_i is a set of i's neighboring agents, and d_a is the degree of agent a. α_i is a ratio of i's degree and the maximum degree in the network. α_i is used to weight i's opinion (i.e., the observer's) since α_i represents i's position in the network. It

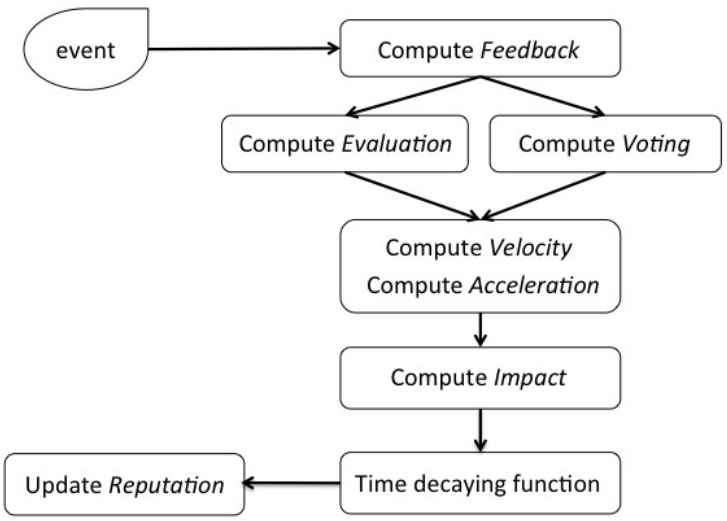


Fig. 4.1: The sequence of processes agents take to update reputation values.

implies that we can infer an agent's confidence from its relative degree information. Given i and it's neighbor l, β_{il} is a ratio of l's degree and the sum of i's neighbors' degrees. β_{il} represents l's relative credibility from i's perspective. i will use β_{il} to collect opinions from its neighbors. The neighbors of each l will recursively compute β_{l*} in turn until one of the three terminating conditions is met as explained in Section 4.2.2. In the voting algorithm shown in 1, i evaluates voting weights for each of it's neighbor l.

4.2.2 Compute Feedback

Feedback process consists of two subprocesses, *Evaluation* and *Voting*. *Feedback* is a part of reputation updating function in Section 4.2.6.

Compute Evaluation.

After each interaction, agents that were involved in the interaction evaluate the interaction according to their own evaluation methods. Therefore, *Evaluation* of interactions is subjective and can be implemented depending on applications. $Evaluation$ of an event e is represented as a function, $Eval(e)$.

Compute Voting.

While the evaluation function is computed by each agent, agents collect opinions from other agents before updating the reputation of the target agent through a voting process to combine diverse opinions. If a_i had an interaction with a_l, a_i can take a vote about a_l to its neighbors to obtain more objective views. The neighbors of a_i can either return a vote to a_i with their own evaluations (if they don't have neighbors other than a_i or a_l) or they can spread the vote to their neighbors. $V^e_{a_i a_l}$ is the weighted sum of voting results and we define it as follows.

$$V_{il}^{e} = \sum_{k \in N_i} \beta_{ik} \times F_{kl}^{e} \tag{1}$$

β_{ik} represents i's delegation trust towards k and is multiplied by F_{kl}^{e} which is a weighted sum of i's evaluation of l on the event e and collected feedbacks from k's neighbors about l. $Feedback$ process is represented with the function F_{il}^{e} which recursively calls the voting function, V_{il}^{e}.

$$F_{il}^{e} = \alpha_i \times Eval_i(e) + (1 - \alpha_i) \times V_{il}^{e} \tag{2}$$

α_i implies self-confidence of i and it is multiplied by $Eval_i(e)$, the self evaluation on event e.

As shown in formula (1) and (2), *Feedback* and *Voting* processes are defined recursively.

Stoping Criteria.

To avoid infinite loops or circular voting processes, we need to specify a few restrictions. First, when an agent takes a vote to its neighbors, it excludes itself. Since the agent's opinion about the target agent is already included in the evaluation function, it only needs to hear from its neighbors. Second, for the voters to avoid casting duplicate votes, each agent keeps history of votes which it has already participated. This is beneficial to the voters so that they don't waste their own resources on duplicate votes. Third, the base condition of the recursive voting is: (1) when an agent has only one neighbor which originated the voting process, (2) when an agent has two neighbors one being the originator and the other being the target agent and (3) when an agent has already participated in the current vote. In the first two cases, (1) and (2), the voter agent returns its reputation value of the target agent.

4.2.3 Compute Velocity and Acceleration

We also consider the frequency of events to compute reputation since an event with a dormant period should be treated differently from frequent ones. We define the velocity for each event to compute the acceleration of events. Then the acceleration of an event influences *Feedback* value of the event through the *Impact function*.

Velocity of an event is defined as follows.

$$Vel(e) = \frac{1}{t_e - t_{e'}} \tag{4}$$

where $t_{e'}$ is the time of the last event.

It is obvious that there needs to be two events to compute velocity, otherwise the velocity is zero. Also, since we consider time with increasing positive integers, $t_e - t_{e'} > 0$ and $Vel(e) \in [0, 1]$.

Now, we can compute the acceleration of event e to identify if its velocity is increasing or decreasing through $Acc(e)$. Since we define the distance between two events are always 1, regardless of the time difference, the acceleration between two consecutive events, e and e', is defined as the change of velocity [18].

$$Acc(e) = Vel(e) - Vel(e') \tag{5}$$

4.2.4 Compute Impact

We introduce *Impact function* to calculate the influence of $Acc(e)$, defined in (5), on the feedback, F^e.

$$I(Acc(e), F^e) = |Acc(e)| \times F^{e^{3^{\frac{-Acc(e)}{|Acc(e)|}}}} + (1 - |Acc(e)|) \times F^e \tag{6}$$

The magnitude of $Acc(e)$ determines the curve of the function which decides how much

to increase or decrease from F^e. When $Acc(e) > 0$, I increases the original feedback value, $I(Acc(e), F^e) > F^e$, and when $Acc(e) < 0$, I decreases the original feedback value, $I(Acc(e), F^e) < F^e$. If $Acc(e) = 0$ then $I(Acc(e), F^e) = F^e$ which means that when there is no change in the velocity, no impact is made to the feedback value, F^e.

4.2.5 Time Decaying Function

Time decaying function is an essential part of the reputation computation since it captures the temporal nature of information; old reputation value may not be as accurate as a new one. Intuitively, an interaction shortly after the previous one can make more use of the built-up reputation (current reputation) while an interaction after a long inactive period should rely more on the current feedback values since the built-up reputation is not up to date. As discussed in [19], time decaying function should to be designed carefully, based on the context (e.g. a periodic function) so that it can adjust time sensitivity weights when computing reputations.

We use an exponential decay function to capture the idea. Our time decaying function relies on the elapsed time since the last interaction.

$$D(x) = e^{-x}$$

where x is $t_e - t_{e'}$.

4.2.6 Update Reputation

Finally, we are now ready to explain the reputation update function that utilizes the functions discussed so far. A new reputation value is computed when a new event occurs. The new reputation is a weighted sum of the current reputation and the feedbacks. The current reputation is weighted by the time decaying function and *Impact function* is applied to the feedbacks. Finally, we formally define the reputation update function as follows.

Reputation update function:

$$R_{il}^{t_e} = d \times R_{il}^{t_{e'}} + (1 - d) \times I(Acc(e), F_{il}^{e})$$

where $d = D(t_e - t_{e'})$.

4.2.7 Ask Function

In our algorithm, each agent keeps a list of reputation values of the neighbors. However, in some occasions, an agent might wonder about another agent's reputation other than the neighbors. Therefore, we implement *Ask function* to query a target agent's reputation who is not a neighbor. *Ask function* is the same as *Feedback* function except the agent does not have its own evaluation of the target agent. *Ask function*, then, goes through the same processes as in *Voting* function as in (1).

$$Ask_{il}^{t} = \begin{cases} R_{kl}^{t} & l \in N_k \\ \sum_{k \in N_i} \beta_{ik} \times Ask_{kl}^{t} & \text{otherwise} \end{cases}$$

4.3 Experiments

In this section, we present two sets of experiments. First, we compute reputations of Autonomous Systems (AS) in the Internet using our algorithm. And we compare our results with *AS-CRED* [20], which is a well-known reputation service for AS network. We use subsets of the real AS networks obtained from *RouteViews* [21]. Second, we study the emergence and propagation of reputations within random social networks generated by Graph-Generator [22]. Graph-Generator is a small Java program to create random graphs in which the number of nodes, edges and the maximum degree are specified.

4.3.1 Experiment 1: Computing Reputations of Autonomous Systems

We apply *ReMSA* algorithm to compute reputation values of Autonomous Systems. An Autonomous System is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators. Since behaviors of each AS represent human interests, we consider AS network as social network. We analyze Border Gateway Protocol (BGP) updates data, which is the standard communication protocol for interconnecting ASes, to evaluate validity of activities among ASes in the network and compute reputation values based on the evaluations. The reputations of ASes could be incorporated for the routes deciding algorithm for each AS since reputations of ASes directly represent the behaviors of ASes.

Network Sampling Algorithm.

As shown in Table 4.1, the number of nodes in the original AS network is very large because it represents all ASes in the Internet. However, only less than 10% of ASes appear in each day's BGP update data we use. Therefore, for the tractability of the experiments, we extract two representative, scaled down, sub-networks from the original AS network. If we sample sub-networks from the original AS network using existing algorithms, most of the ASes in sampled sub-networks don't appear in the BGP data. Instead of using the algorithms discussed in [23], we designed a context-based network extraction algorithm in order to sample meaningful sub-networks which contain most of the autonomous systems appearing in BGP update data. Therefore, we extract ASes that appeared in the BGP data so that we can compute reputations of the ASes. For each randomly chosen *ASPATH* in BGP update data on January 1, 2010, we add ASes which appear in the *ASPATH* to the sub-network and repeat the process until the desired number of nodes for the sub-network is reached (in this case, 5000).

In order to measure whether the sub-networks represent the original AS network rea-

sonably, we evaluate the sub-networks by the metrics defined in [23].

	Original	sub-network1	sub-network2
# Nodes	33508	5000	5000
# Edges	75001	19953	22379

Table 4.1: Nodes and edges information of networks.

	in-deg	hops	sng-val	sng-vec	clust	AVG
sub-network1	0.2743	0.3500	0.1883	0.1180	0.2346	0.2399
sub-network2	0.0703	0.3500	0.1234	0.0357	0.1944	0.1547

Table 4.2: Sampling criteria for sub-networks.

Table 4.1 shows the number of nodes and edges of the original network and two sampled sub-networks. The number of nodes of sub-networks (5000) is approximately 15% of the real network (33508) which is enough to match the properties shown in Table 4.2 [23]. Table 4.2 shows five different distributions of two sample networks measured using *Kolmogorov-Smirnov D-statistics*. *D-statistic* measures the agreement between the true and the sample network property [23]. In the last column, we show the average of the results. Lower average values means more agreement between original network and the sub-network. Some good average values discussed in [23] were 0.202 and 0.205. Therefore, both of the sub-networks qualify for scaled-down sub-networks well representing the original.

Evaluation of BGP.

We use BGP (Border Gateway Protocol) update data from *RouteViews* [21] dated from January, 2010. Also, we use the same analysis of AS-behavior discussed in [20] and [24] to compute feedbacks of BGP activities in order to compare our reputation computation results with *AS-CRED*. In order to use our algorithm, we need to define *events* and associated *observer* and *observee* in the problem domain of AS reputation computation. In BGP update data, for each update message sent from say, $AS0$ to $AS1$, $AS1$ analyzes the message

as an observer and evaluates $AS0$'s behavior. Such a message is an event as we defined in Section 4.2.1. And an event can be a non-empty subset of the set *{AS-Prefix behavior, AS-Path behavior, AS-Link behavior}*, which represents the *observee's* behaviors. Each behavior can be evaluated to be positive or negative and then the result of the evaluation is accumulated for that message. In other words, each message will have an associated score representing the behavior of the *observee.* We describe how each behavior is evaluated below.

- *AS-Prefix behavior*: For the observee AS and its prefix p, we compute two temporal metrics, *persistence and prevalence.* These two metrics can represent positive or negative behavior of the observee AS. We will not discuss the details of the metrics because they are beyond the scope of this paper. The value of the persistence and prevalence are compared against a set of thresholds mentioned in [24] and feedback is provided. For good behaviors, evaluation of 1 is provided and otherwise -1.

- *AS-Path behavior*: We use AS relationship data from [25] to evaluate the valley free property of AS $paths$. None of the ASes in the AS $path$ should form a valley. The observee AS provides an AS-Path. If a valley is found in the provided AS-Path and the first AS forming the valley is the observee, it gets evaluated by its observer with -1.

- *AS-Link behavior*: For each link in the AS-Path provided by the observee, we compute persistence and prevalence values, then these are compared with the threshold discussed in [24]. If the classification of the behavior is good, an evaluation of 1 is provided, otherwise the link is unstable, therefore, the evaluation is -1.

Results.

We compute reputations for each AS appeared in BGP update data for each day between January 1, 2010 and January 31, 2010. We average the reputations computed by *ReMSA*

with two different sub-networks. We exclude ASes with good behaviors (positive reputation values) to compare the result with *AS-CRED* which accumulates reputation values when bad behavior occurs (zero is considered the best reputation in *AS-CRED*).

In Figure 4.2, we show the distribution of reputation values of ASes computed by *AS-CRED* for January 1, 2010. In Figure 4.3, we show the distribution of reputation values of ASes compute by *ReMSA* for the same day. The ranges of reputation values shown in the figures are different as they represent the raw reputation values computed by *AS-CRED* and *ReMSA*. Since *AS-CRED* computes centralized reputation values, we averaged reputation values computed for each AS by *ReMSA*. The purpose of each algorithm is implied by the distribution of reputation values shown in the figures. *AS-CRED* computes reputation values of ASes to detect globally malicious ASes while *ReMSA* computes distributed reputation values for each AS to measure the trustworthiness of relationships between neighbors. Therefore, *ReMSA* allows each AS to have its private perception of its neighbors based on the history of interactions and the witness information rather than to rely on global computed values.

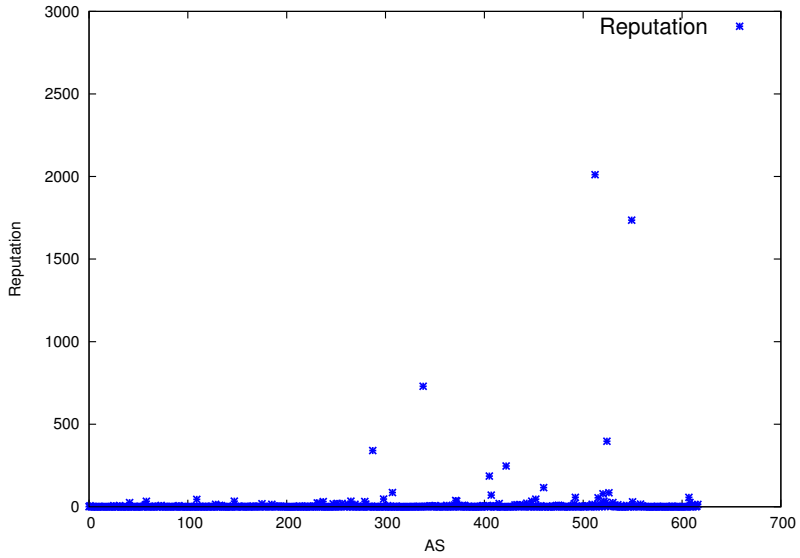


Fig. 4.2: Reputations computed by *AS-CRED*.

Figure 4.4 shows average reputation values of ASes computed by *AS-CRED* and *ReMSA*

over one month. The non-zero reputation values of ASes are added and averaged from our sub-networks. Similarly, the reputation values of respective nodes from *AS-CRED* were averaged. We normalized values computed from *AS-CRED* since zero is the best reputation value and the higher values represent the worse reputation in their algorithm.

The two lines from *AS-CRED* differs in that the one below doesn't include *AS209*, which has an extremely bad reputation with the raw value 2755.52. We investigated the differences shown in Figure 4.4 and found out that whenever there are big gaps, e.g., on the 14th day, there was an AS with extremely bad reputation (i.e. $AS209$) in *AS-CRED*. Therefore, when we normalized the reputation values, since the worst reputation value in *AS-CRED* becomes -1, it makes other reputation values negligibly small. Consequently, the normalized average for *AS-CRED* is smaller than our algorithm's average. For example, on the 14th day, $AS209$ received an extremely bad reputation (the raw value 2755.52) when most of other ASes received less than 10. Such a huge difference among reputation values makes other reputation values negligible which enables moderately malicious ASes to get hidden under an extremely malicious AS.

Now let's observe *AS209* more closely. Figure 4.5 shows the reputation value of $AS209$

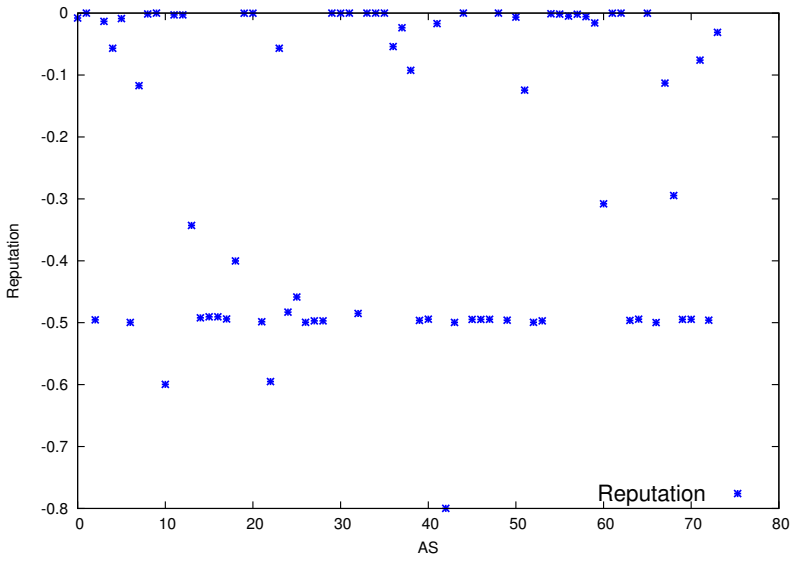


Fig. 4.3: Reputations computed by *ReMSA*.

computed by *AS-CRED* and our algorithm. In the figure, we normalized reputation values from *AS-CRED* with the largest value $AS209$ had, which was on January 14th, 2010. $AS209$ had bad behaviors before the 14th, but because of the huge variances among the reputation values over time in *AS-CRED*, the reputation values of $AS209$ on other days except the 14th became almost zero after normalization. *AS-CRED* may be useful to identify the most malicious AS, but it may lose other important information such as how reputation value changes over time.

We also compare the reputation ranking sequences of ASes by *AS-CRED* and *ReMSA*. For each day in January, 2010, we order ASes by their reputation values computed by *AS-CRED* and *ReMSA* respectively. Then, we compare the two sequences of the orderings by calculating standard deviations of the sequences. Figure 4.6 shows average standard deviations for each day. Average standard deviations vary each day slightly, but the value keeps low compared to the maximum value which is 2500.

In Figure 4.7, we show sample reputation rankings computed from *AS-CRED* and *ReMSA*. We randomly picked 100 ASes to compare average rankings computed throughout a month, January, 2010. The rankings computed by *ReMSA* is ordered in increasing order

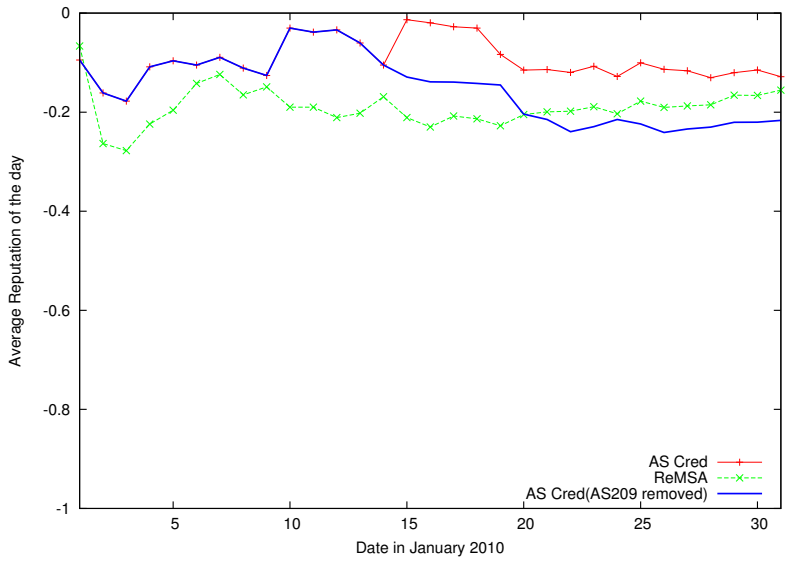


Fig. 4.4: Reputations computed by *AS-CRED* and *ReMSA*.

and rankings by *AS-CRED* is presented accordingly. Again, we can observe that the overall ranking trend agrees with minor exceptions.

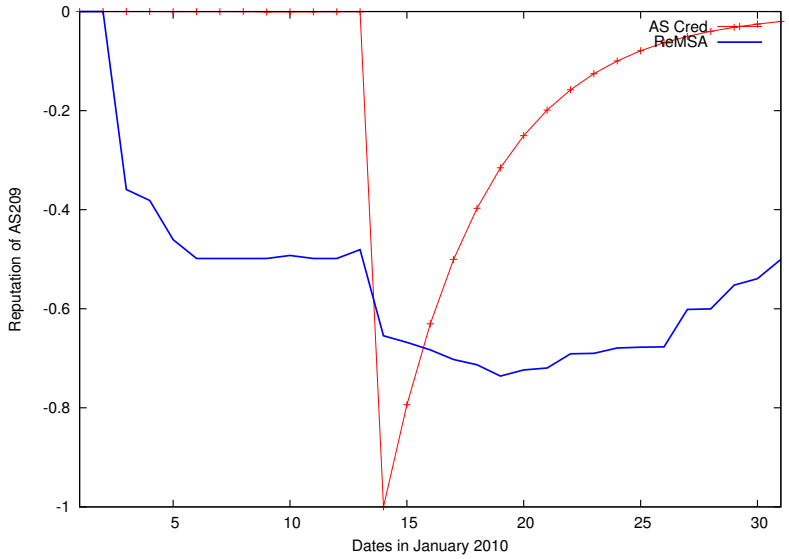


Fig. 4.5: Reputation values computed by *AS-CRED* and *ReMSA* for $AS209$.

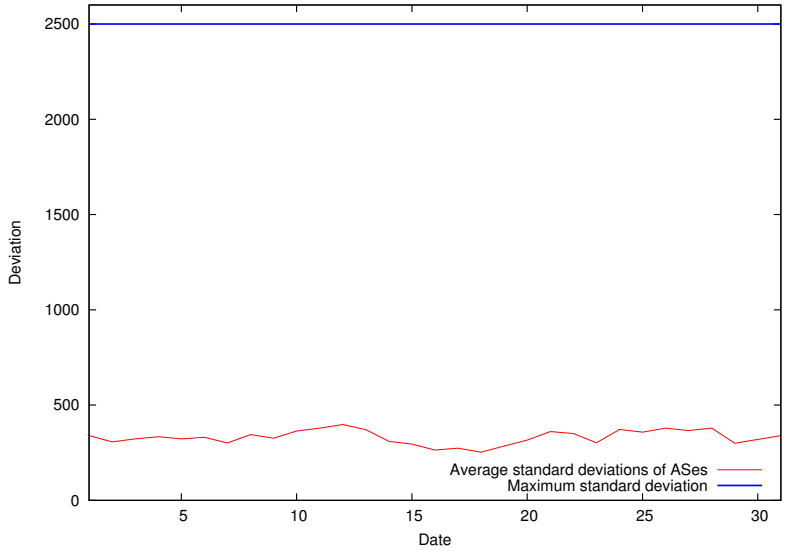


Fig. 4.6: Average standard deviations of reputation rankings of ASes computed by *AS-CRED* and *ReMSA*. The perfect standard deviation is 0 and the maximum standard deviation is 2500.

4.3.2 Experiment 2: Propagation of Reputation on Random Networks

In addition to the performance evaluations on a real social network (ASN) presented in Section 4.3.1, we test the propagation of reputation values in random networks. We study a *sparse* and a*dense* network in order to show how topology of the networks (degrees of nodes) influence propagation of information (reputation values). Table 4.3 shows the statistics of the two random networks.

	Sparse network	Dense network
# Nodes	5000	5000
# Edges	10000	50000
Average Degree	4	20
Density	0.001	0.004
Diameter	11	4
Average Path Length	6.624	3.129

Table 4.3: Statistics of two random networks.

For each network, we pick an observee node, a_0, and observe how reputation computed by its neighbors change over time. We also pick two observers distant from a_0, say A and B,

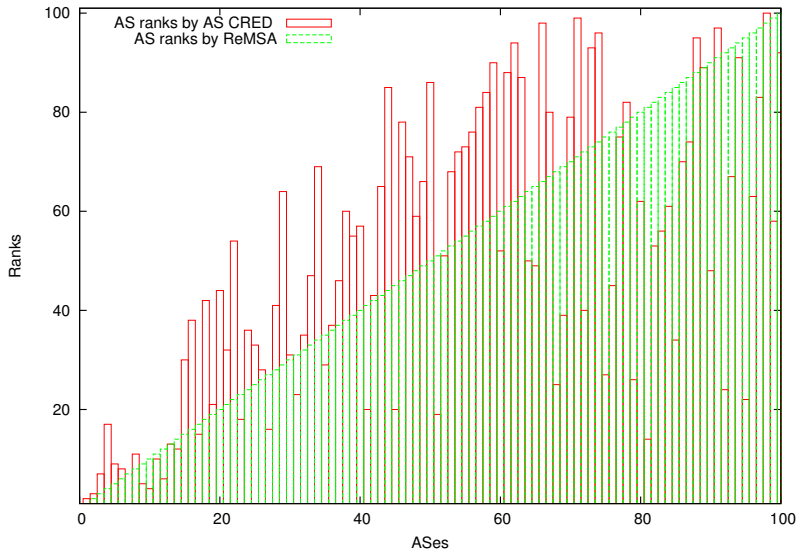


Fig. 4.7: Sample average reputation rankings computed by *AS-CRED* and *ReMSA*.

in order to show a_0's reputation values obtained by each observer. Note that each observer will obtain a subjective reputation value about a_0. We generated random events that are associated with time information, an observee node and the evaluation of event. Each node has behavioral probability (positive or negative) and the observer evaluates behaviors of observee nodes.

The straight line in Figure 4.8 shows the true behavior of a_0 based on its behavioral probability, p_{a_0}. We define p_{a_0} as the probability that a_0's behavior is evaluated to 1. In this case, the probability was 0.1 and therefore the true reputation (true reputation = $2 * p_{a_0} - 1$) is -0.8 since the reputation value is between -1 (when p_{a_0}= 0) and 1(when $p_{a_0} = 1$). The neighbors have interactions with a_0 and update reputations based on the probabilistic random behaviors of a_0. The average reputation values of a_0 computed by the neighbors lie right above the true reputation value in Figure 4.8. The two other lines on top represent the reputations values seen by A and B. For each time the neighbors' reputation values of a_0 are computed, A and B query reputation of a_0 using Ask function presented in Section 4.2.7. As we can see in Figure 4.8, it is not hard to believe that direct interactions with a_0 help compute more accurate reputation values of a_0 compared to the reputation values received only by collecting opinions from other nodes. Also we can see that the changes in reputation values become more stable as nodes learn a_0's true behaviors and share subjective reputation values of a_0 through voting processes. Since A is 4-hop-away from a_0 and B is 6-hop-away form a_0, we can see that A has closer values to the true value than B.

We repeat the process on the dense network and the results are shown in Figure 4.9. We set the behavioral probability of the observee, say a_1, the same. A and B both were 3-hop-away from a_1. The average reputation values computed by a_1's neighbors converge closer to the true value. On the dense network, reputation values seen by the two observers fluctuate because, as shown in Table 4.3, the network is so dense and the reputation of the target agent is influenced by many different agents through *Ask Function*.

In Figure 4.10, we show how velocity of events influence reputation values. As dis-

cussed in Section 4.2.4, the *Impact function* adjusts the computed feedback values based on the acceleration of the event. On a random network, we pick a node with behavioral probability 0.8 and observe reputation values from a neighbor changing over time when the velocity of interactions is constant and when the velocity of interactions increases. As we can see in Figure 4.10, the reputation computed without acceleration becomes stable as it reaches close to the node's true reputation, 0.6, while the reputation computed with

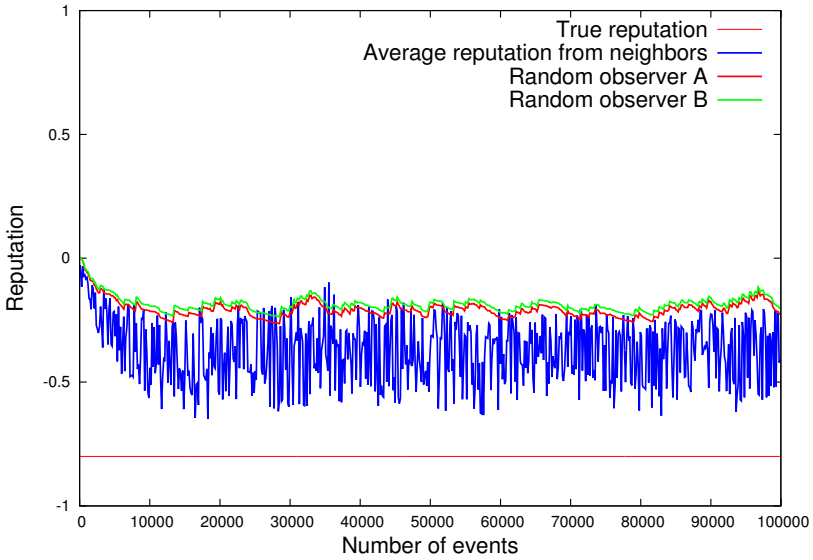


Fig. 4.8: Propagation of reputation in a sparse network.

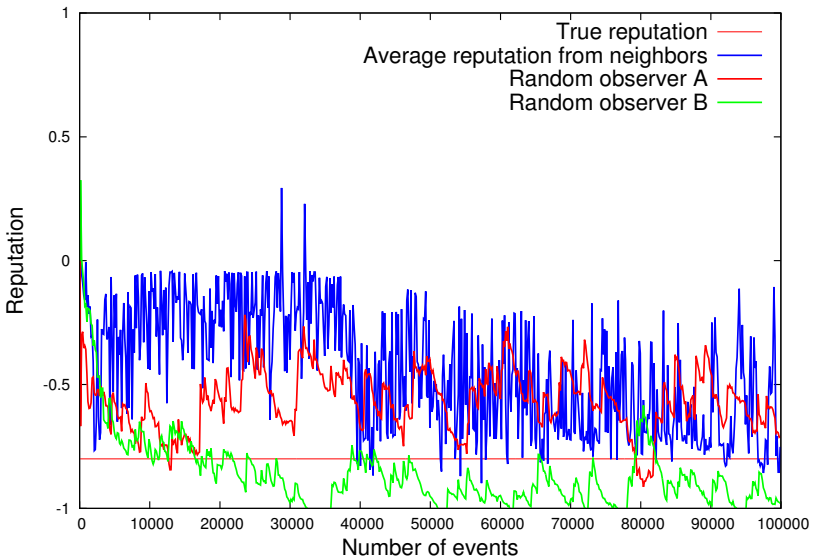


Fig. 4.9: Propagation of reputation in a dense network.

nonzero accelerations fluctuates more. Since the *Impact function* emphasizes the influence of accelerations of events, the reputation values become more sensitive to the current feedbacks when the rate of events is more frequent.

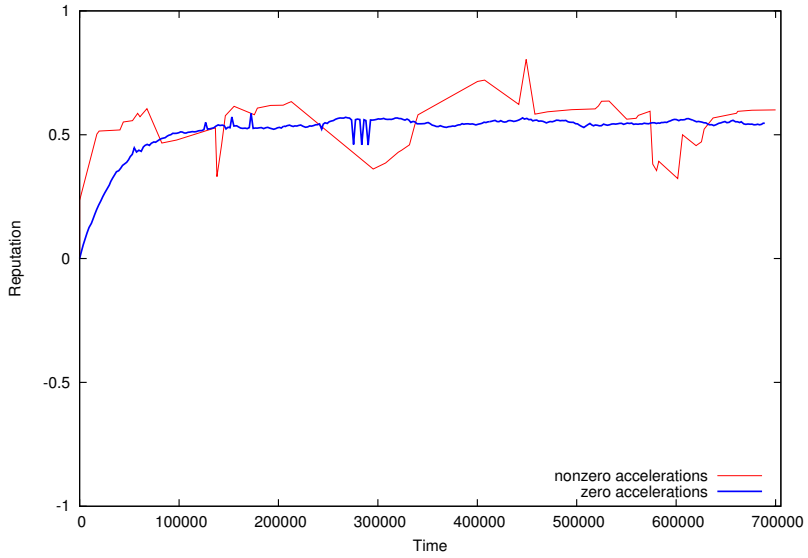


Fig. 4.10: Reputations computed with constant and increasing velocity.

4.3.3 Effects of Voting, Time Decaying Function and Impact Function

In this Section, we show the effect of each mechanism in the reputation update formula, introduced in Section 4.2.6. We create a scale-free network using *Barabási-Albert* algorithm with 5000 nodes and 50000 edges. This network was used in the following experiments. We discuss the results of experiments using a sample agent picked randomly which represents the typical behavior of agents in general. The reputation values are computed from the neighbors of the sample agent.

Effects of Voting.

Voting process is introduced in Section 4.2.2. Through the voting, one can aggregate others' opinion so that the computed reputation values are objective. The balance between

direct experiences (self opinion) and indirect experiences (others' opinions) is automatically controlled by the degree of each agent as discussed in Section 4.2.2.

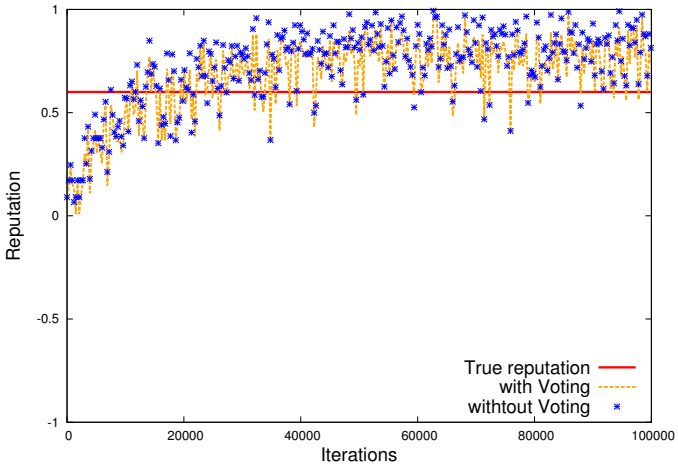


Fig. 4.11: Effects of Voting

Node	1453	4999	3387	4102
Degree	10	11	33	57
With voting	0.247	0.187	0.156	0.142
Without voting	0.311	0.291	0.234	0.175

Table 4.4: Average distance from the true reputation.

Figure 4.11 shows the reputation computed with and without the voting process which means that the *feedback* function is replaced by *Evaluation* (self opinion) only. The straight line is the true reputation of an agent. The reputation values computed with voting, which considers others' opinions, are closer to the true reputation value compared to the reputation values computed without voting. Table 4.4 shows the average distance from the true reputation value over the iterations for four sample nodes. The average distance from the true reputation is lower when the reputation values are computed with voting. We also observe that as the degrees of node increases, the average distance from the true reputation decreases since having more neighbors lead to getting more opinions.

Effects of Time Decaying Function.

The time decaying function, discussed in Section 4.2.5, utilizes the frequencies of interactions to balance the current reputation and the feedbacks. In some applications, the steady-state reputation is more valuable , while in other cases, reputation values need to be adaptive so that they reflect the up-to-date information. In *ReMSA*, this is automatically controlled by the time decaying function.

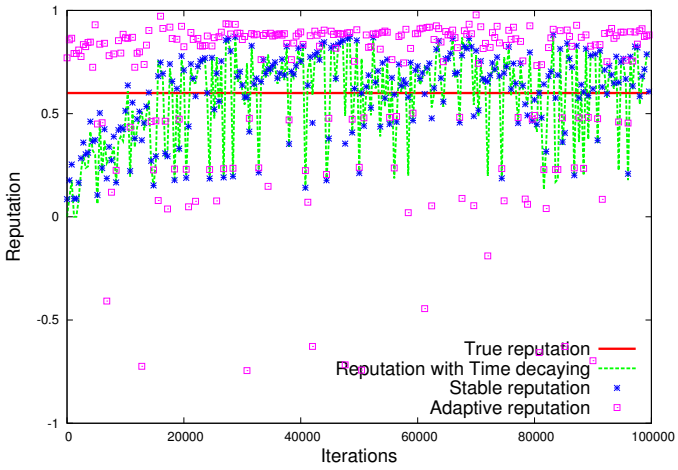


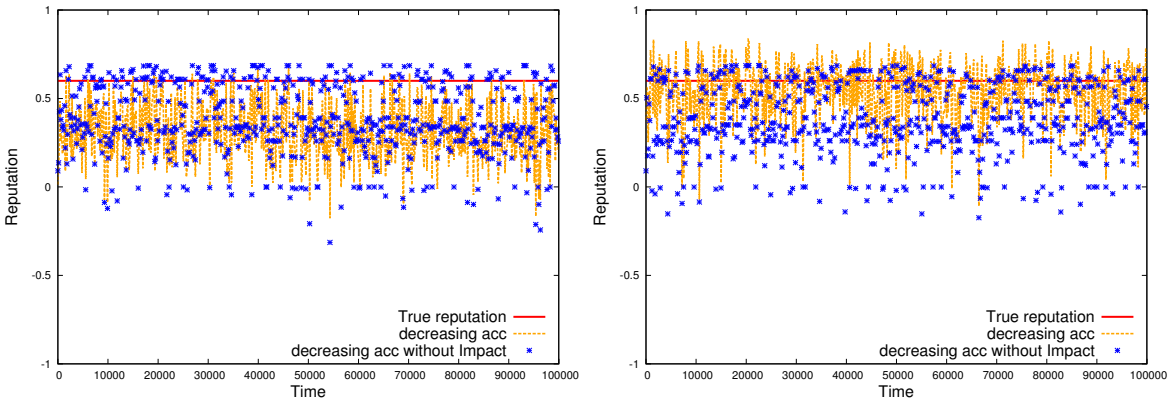
Fig. 4.12: Effects of Time decaying function.

Figure 4.12 shows four different reputation values. The straight line shows the true reputation value of the agent under observation. The line shows reputation values computed using standard *ReMSA*. We also show steady-state reputation values as well as adaptive reputation values, plotted with square points and star points respectively. As discussed before, new reputation is computed as a weighted sum of current reputation and new feedback. Steady-state reputation has more weight on current reputation while adaptive reputation has more weight on new feedback. For the comparison, we weight current reputation with 0.9 and new feedback with 0.1 for steady-state reputation and vice versa for adaptive reputation. Intuitively, if the current reputation is weighted more than the new feedback, the reputation value is stable meaning it does not fluctuate much. On the other hand, if the new feedback is weighted more than the current reputation, the reputation value is more adaptive since the updated reputation value reflects more of the current behavior of an agent

than the history of the past behaviors. As shown in the Figure 4.12, adaptive reputation values are distributed near 1, 0, or -1, which are the raw feedback values. Steady-state reputation values are distributed near reputation values computed by standard *ReMSA*; this is because the interactions are randomly generated and the frequencies of interactions do not vary much.

Effects of Impact Function.

The purpose of *Impact function* is to reflect accelerations of repeated interactions to the *feedback*. In real social networks, repeated interactions in a short period of time may imply greater closeness of two entities involved. Therefore, we emphasize sudden increases of interactions rate using *Impact function*. In Figure 4.13(a), reputation values of interaction with increasing accelerations are shown and in (b), reputation values of interactions with decreasing accelerations are shown. Since the time information of the interactions are randomly generated integers, the acceleration values are small. For example, if three interactions with time 10, 20, 25 occurs, the velocity of the second and third interactions are 0.1 and 0.2 and the acceleration of the third interaction is 0.1. Therefore, even the acceleration of the third interaction is positive and emphasized by *Impact function*, the difference is not big. With positive accelerations, positive *feedback* values are rewarded and negative *feedback* values are punished according to the acceleration. In Figure 4.13(a), reputation values below the red line show that reputation values computed with *Impact function* are lower than the ones computed without *Impact function*. Since the acceleration is positive, negative feedbacks were punished (decreased) by *Impact function*. On the other hand, in Figure 4.13(b), reputation values of interactions with negative accelerations are shown. Reputation values below the red line shows that reputation values computed with *Impact function* are higher than the ones computed without *Impact function* since negative *feedbacks* with decreasing accelerations are rewarded (increased) according to the acceleration values.



(a) (b)

Fig. 4.13: Effects of Impact function.

4.4 Chapter Summary

A natural way of measuring an agent's reputation is to accumulate interaction history in some form of weighted sum. We started with this idea and incorporated frequency and velocity of interactions as well as the topological information of the network. Generally, reputation computation takes an approach that combines the previous reputation and new experience.

Our algorithm assumes that each agent is aware of its neighbors and its position in the network and makes use of the information. Therefore if the topology of network changes, an agent perceives its new sociological information and its reputation updating function changes accordingly. This is explained by human behavior; people tend to act with more responsibly when there are more observers (i.e., a higher degree). Instead of using neighbor's reputation value as a weight (as in *HISTOS*), we take advantage of neighbor's relative degree as a weight which is more objective. Also, in our algorithm, when an agent takes a vote of its neighbors, the neighbors can recursively take votes of their neighbors.

Through the experiments, we show that our algorithm can compute quality reputation values by comparing the results with an existing reputation computation algorithm (*AS-CRED*). The algorithm can also successfully model propagation of reputations within the

network. We show that in a dense network, reputations travel much quicker and diffuse wider given the same number of interactions. In addition, we show how frequencies of interactions can influence the reputation values. Since a higher rate of interactions implies greater significance, we believe that the velocity of interactions is an important parameter in our algorithm.

Since *ReMSA* is designed for distributed environments, the algorithm is employed within each agent. Therefore the time complexity of *ReMSA* for each agent is dependent on the number of events and the number of neighbors each agent has. Then the time complexity of the algorithm is $O(d_i * |E_i|)$ for each agent i.

CHAPTER 5
DISTRIBUTED SECOND DEGREE ESTIMATION IN SOCIAL NETWORKS

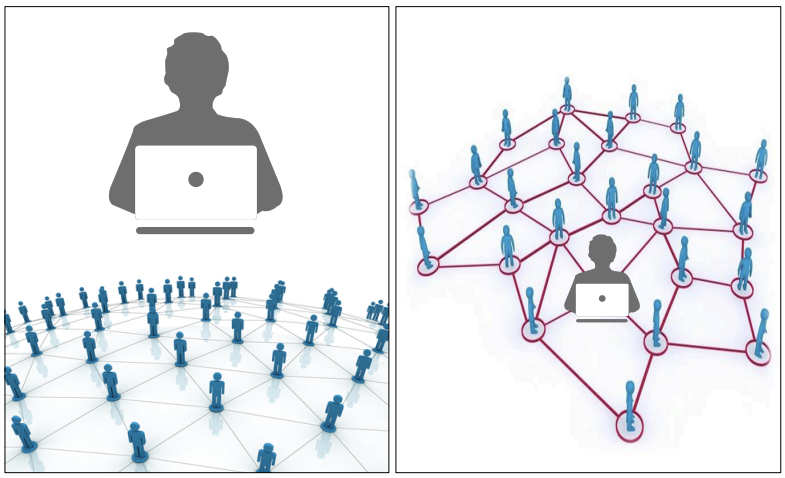
In this Chapter, we propose an agent centric algorithm that each agent (i.e., node) in a social network can use to estimate each of its neighbor's degree. The knowledge about the degrees of neighboring nodes is useful for many existing algorithms in social networks studies. For example, algorithms to estimate the diffusion rate of information spread need such information. In many studies, either such degree information is assumed to be available or an overall probabilistic distribution of degrees of nodes is presumed. Furthermore, most of these existing algorithms facilitate a macro-level analysis assuming the entire network is available to the researcher although sampling may be required due to the size of the network. In this paper, we consider the case that the network topology is unknown to individual nodes and therefore each node must estimate the degrees of its neighbors. In estimating the degrees, the algorithm correlates observable activities of neighbors to Bernoulli trials and utilize a power-law distribution to infer unobservable activities. Our algorithm was able to estimate the neighbors' degrees in 92% accuracy for the 60867 number of nodes. We evaluate the mean squared error of accuracy for the proposed algorithm on a real and a synthetic networks.

5.1 Motivations

As online social networks gained significant popularity, understanding the characteristics of social networks became an essential task for many application areas. Many existing studies in social networks uses a macro-level approach to a variety of problems including diffusion of influence, malicious node detection, and efficiency of communication networks [26]. Recent research interest has been shifting to a micro-level or a node-level reasoning. These studies focus on designing algorithms for an individual node within a social network to reason about the characteristics of the social network it belongs. Due to privacy protection in social networks and their dynamic characteristics and extremely large sizes, such node-level reasoning is extremely challenging.

Among many properties of social networks, researchers accept the *degree distribution* of a network as the most essential property in understanding the structure of the network. The degree distribution $P(k)$ of a network is the fraction of nodes in the network with degree k. Therefore, many have realized the importance of reasoning about degree distributions in social networks and studied it in-depth [27]. However, as alluded by the definition, the knowledge of degree distribution does not provide information on the degree of any arbitrarily chosen node; it merely gives a probability value for each possible degree. The degree of a node represents the importance, or influential power, of the node because a higher-level of connections implies a higher level of diffusion may the node facilitate. If a node knows the degree of a neighbor, it may infer to the popularity of the neighbor [28]. Attempts to estimate degree distributions of networks have been made by, but not limited to, [26], [29], and [30].

However, most researchers have employed a macro-level approach, such as graph sampling, to estimate degrees of nodes in social networks. In a distributed reputation management knowledge of the second degrees–i.e., the sum of degrees of neighbors–is important [17]. Figure 5.1 presents the difference between macro and micro-level approaches. In a macro-level approach, shown in Figure 5.1 (a), also referred to as a global method,



(a) Macro-level approach (b) Micro-level approach

Fig. 5.1: The difference between macro and micro level approaches to social network analysis. In general, a macro-level analysis is conducted offline using a collected data set, while a micro-level is an online analysis that can continuously update the belief about the world.

there is an entity outside of a given network that can observe everything about the network such as the number of users, degrees, etc. Therefore, the goals of a macro-level approach is often to analyze structure of the network and learn interesting properties of the network, from an objective view with offline data sets collected. On the other hand, in a micro-level approach, shown in Figure 5.1 (b), also referred to as a distributed method, the observing entity is a member of the network and therefore it can only access private information, such as its own degree.

Usually, a micro-level approach can be used for an online algorithm capable of updating its knowledge over time. For example, on a sensor network, where each node itself is a tiny computer, it is infeasible for each node to have global view of the entire system at any time [31]. Therefore, in distributed systems like sensor networks, tasks are solved completely locally by each node running distributed (node-centric) algorithms. One the other hand, a macro-level approach generally requires an offline data set that is sampled for a given period of time.

In addition, as social media marketing proved itself to be powerful, marketers are increasingly investing in blogging and social media. They are interested in finding the most influential users in social media so that they can achieve the maximum efficacy. Since the influential power of a user is generally represented by the number of connections (degrees) of the user [32], and the marketers usually do not have information about topology of social media, estimating degrees of users in micro-level is recommended.

In this Chapter, we introduce a node-centric, i.e., micro-level, algorithm to estimate degrees of neighboring nodes.

5.2 Previous Works on Degree Estimations

Estimating the degree distribution is the first step toward understanding the nature of networks according to [26]. The authors describe why the degree distribution of a social network may not be public knowledge and proposes an algorithm to estimate it from a database while keeping the privacy of the information. Their algorithm is a macro-level algorithm to approximate an overall probabilistic degree distribution of a network as defined above. Also an important assumption of their approach is that databases are available to query. Our algorithm does not assume the availability of a database containing information about the network. Furthermore, instead of estimating an overall distribution of the degrees in a network, our algorithm is to be used by a node within a network to reason about degrees of its neighbors. Using the preferential attachment model [33] and our algorithm, a node within a social network can reason about the entire network it belongs. This paper focuses on estimating neighbors' degrees. We make a reasonable assumption that each node knows the degree of itself, i.e., the number of neighbors. For example, in the *Facebook* network, each user knows exactly how many *friends* (neighbors) he or she has but the user does not always know how many *friends* his/her *friends* have. In an online social network, each node is interested in knowing the degrees of its neighbors for various reasons such

as to find the most influential neighbor to spread information in the network and to select a neighbor who is most likely to share quality information. Another practical example is on sensor networks. A node in a sensor network can choose a neighbor that may have the highest degree among its neighbors to transfer/spread information it gathered to the entire network as fast as it can. From the best of our knowledge, no attempt has been made to estimate degrees of neighboring nodes in a distributed manner.

In [30], authors attempt to estimate the degree distributions of a network using different sampling algorithms including random walks; this is again a macro-level approach. [34] introduces three estimators, namely, maximum likelihood estimator, mark and recapture, and random walker, to estimate the size of a social network. The second degree of a node is the sum of all degrees of the node's neighbors as defined in [28]. They also suggest that the distribution of second degrees is of interest since it is a good approximation of PageRank [35]. They prove that the distribution of second degrees follow a power law in *Buckley-Osthus* model which is a generalization of the *Barabási-Albert* model.

Most of the existing researches focus on the topology of given networks and studies the degree (or second degree) distributions based only on the topological information. However, to be able to deal with real world networks, in which nodes and edges are dynamically changing, one cannot assume that the topology of networks are known in advance and, therefore, the presented methods above are not suitable for distributed and dynamic environment.

5.3 An Algorithm for Estimating Each Neighbor's Degree

In this section, we discuss our algorithm and explain how each node within a social network can use the algorithm to estimate the degrees of its neighbors. We first present an overview of the algorithm then present several important definitions to help the readers

to understand the algorithm in detail. Then we present an important proposition with a proof to show that our algorithm indeed can compute accurate degree values if the numbers of observations about its neighbors follow a certain proportionality condition (Definition 5.3.10). We also show that straightforwardly counting the number of observations does not yield good estimations because there are unobservable activities. We use a beta distribution and a power-law distribution models to extrapolate the observed activities to estimating the degrees of neighbors. This idea turned out to be quite effective in discovering hidden neighbors of neighboring nodes as we show in the experiments section.

5.3.1 An Overview of the Algorithm

We assume that the observer node can perceive some activities of its neighbors and we call the observed activities as *observations* (Definition 5.3.5). In reality, a user cannot collect every *observation* of their neighbors. Furthermore, there are nodes that are connected to a node that do not make any observable activities. For example, users of a Facebook wall (a node) may be just reading the posting. In this case, the reading activities are not observable; nevertheless these nodes are still connected to the node and important from the information diffusion or the connectivity concerns. We also assume that, without loss of generality, the differences among the numbers of observations made on neighbors are relative to the degrees of neighbors. If a user can see its neighbor interacting a lot with others, the user can infer the neighbor has a lot of friends (neighbors) compared to another neighbor that has less interactions.

The overall idea of our algorithm is the following. Each node estimates neighbors' degrees based on *observations* made about each neighbor that are proportionally bounded by the current estimation of the total activities – captured by N_v below – between its neighbors and the second neighbors. We consider each *observation* as Bernoulli trial, the number of seen *observations* so far as the number of successes and the number of unseen (expected) *observations* as the number of failures. As a new *observation* is introduced, the success

probability is updated by the beta distribution. The numbers of *observations* made about each neighbor are statistically proportional to each other. Therefore, estimating the neighbors' degrees directly from the number of *observations* without applying to the Bernoulli trials will not capture the statistical properties of the degree distribution of the neighbors. Then, each node adjusts the distribution of second degrees according to a power-law distribution, because previous studies including [36] have shown that degree distributions of social networks follow Power-law distribution. Finally, we apply the principle of maximum likelihood to estimate the degree of each neighbor.

5.3.2 Useful Definitions

We define some useful terms in this Section. Figure 5.2 shows a node v's second neighbors where one of its neighbors is i. We assume that the degree information of a node is only known to itself.

Definition 5.3.1 *Let $deg(i)$ be the true degree of a node i. Then, use $deg_v(i)$ as v's estimation of i's degree. See Figure 5.2.*

Definition 5.3.2 *Let N_v be v's estimated second degree. Then, $N_v = \sum_{i \in Ne_v} deg_v(i)$. See Figure 5.2.*

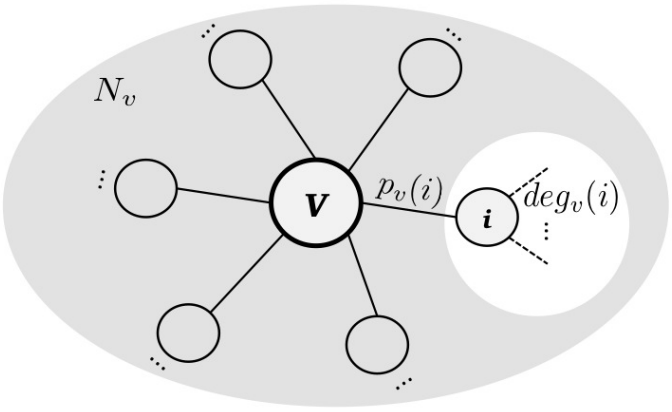


Fig. 5.2: The second neighborhood of a node v and one of its neighbors, i.

Definition 5.3.3 *Consider a network $G =< V, E >$ where V is a set of nodes and E is a set of connections. For a given node, $v \in V$, we denote the neighbors of v as Ne_v.*

Definition 5.3.4 *Given a node v and a neighbor i, let $p_v(i)$ be v's estimated probability that i is connected to a node in each trial according to a binomial distribution.*

We employ *Erdős-Rényi model* [37] to define distribution of degrees of v's neighbors. Under the model, the probability that the degree of i is k given $p_v(i)$ follows a binomial distribution.

According to *Erdős-Rényi model* [37], v's estimated probability that i's degree is k given $p_v(i)$ is defined as follows.

$$Pr_v(deg(i) = k|p_v(i), N_v) = \binom{N_v-2}{k-1} \cdot p_v(i)^{k-1} \cdot (1 - p_v(i))^{n-k-1} \quad (5.1)$$

However, $p_v(i)$ and N_v are not known to v and we explain a method to estimate $p_v(i)$ and N_v using Beta distribution in Definition 5.3.7. Once $p_v(i)$ and N_v are estimated, $p_v(i)$ for each neighbor i is adjusted according to a power-law distribution. Finally, v estimates $deg_v(i)$ using the maximum likelihood principle. A step by step procedure is given in Algorithm 4.

5.3.3 Defining Observations

Generally, in a social network, a node in the network can observe interactions between its neighbors and others. For example, on *Facebook*, a user can observe its *friends'* interactions with others through *wall* postings and *likes*. Intuitively, the degree of a neighbor is correlated to the number of observed and unobserved interactions. We define observed activities as *observations*. An example of an unobserved interaction on *Facebook* includes reading a posting without making any comments.

Definition 5.3.5 (Observation) *Given a node v, an observed interaction between v's neighbor i and i's neighbors is defined as an observation, $o_v(i, t)$, where t is the time of the*

interaction. Also, a time-ordered list of v's observations on i is defined as $O_v(i)$ and O_v is a time-ordered list of $\bigcup_{i \in Ne_v} O_v(i)$.

We use the beta-binomial distribution to update $p_v(i)$ when an *observation* occurs. Since the belief about neighbors' degrees follows a binomial distribution according to [38], we can compute the probability of a neighbor i having a degree k using Equation (5.1). Also, as the beta distribution captures the prior and posterior beliefs of a binomial distribution, each node can update the belief about the degree of each neighbor with the beta distribution considering each *observation* as a binomial trial.

Now we are ready to discuss a method to compute $p_v(i)$ which is needed in Equation 5.1. If the posterior distributions are in the same family as the prior probability distribution, the prior and posterior are then called conjugate distributions, and the prior is called a conjugate prior for the likelihood [39]. In Definition 5.3.6, conjugate prior and posterior distribution are defined when the likelihood function is binomial.

Definition 5.3.6 (Conjugate distributions) *If a prior distribution of $p_v(i)$, v's estimation of the probability that i is connected to an additional node, follows $Beta(a, b)$ and if the likelihood has a binomial distribution, i.e., $f(x|p_v(i)) = \binom{n}{x}(p_v(i))^x(1 - p_v(i))^{n-x}$, the posterior distribution of $p_v(i)$ given x is $Beta(a + x, n + b - x)$.*

Next, we extend the idea from Definition 5.3.6 to compute $p_v(i)$ in Definition 5.3.4, given a node v and its neighbor i.

Definition 5.3.7 *Given a node v, $p_v(i)$ follows $Beta(deg(v), deg(v) + 1)$. Then the estimated posterior distribution of $p_v(i)$ is $Beta(deg(v) + |O_v(i)|, N_v + deg(v) + 1 - |O_v(i)|)$.*

Definition 5.3.7 proposes a method to estimate $p_v(i)$. This process is repeated for each *observation*, $o_v(i, t)$ as described in Algorithm 4. The two parameters for the beta distribution represent the estimated degree of i and the estimated second degree of v, respectively.

Since v is only aware of its own neighbors before any *observation* has been made, v's initial estimation of the second degree is $deg(v) + 1$ (the number of v's neighbors plus itself), as in line 6 of Algorithm 4. Also without any information about neighbors' degrees, initially v assumes for all the neighbors have the same degrees with itself [38], as in line 5 of Algorithm 4. Then, upon each *observation*, v updates $p_v(i)$ with the expected value of the posterior beta distribution and update $deg_v(i)$ with k that gives the maximum likelihood, $Pr_v(\cdot)$ as defined in Equation (5.1), as in line 8-16 in Algorithm 4.

Algorithm 3 An Algorithm for Estimating Neighbors' Degrees

1: **Input:** $O_v = \{o_v(i,t) | i \in Ne_v\}$
2: **Output:** $\{p_v(i), deg_v(i) | i \in Ne_v\}, N_v$
3: **for all** $i \in Ne_v$ **do**
4: $p_v(i) \leftarrow \frac{deg(v)}{2*deg(v)+1}$
5: $deg_v(i) \leftarrow deg(v)$
6: $N_v \leftarrow deg(v) + 1$
7: **end for**
8: **for each** $o_v(i,t) \in O_v$ **do**
9: $vel(o_v(i,t)) \leftarrow \frac{1}{t-t'}$
10: **if** $\int_0^t vel(o_v(i,t))\mathrm{d}t \geq 0$ **then**
11: $p_v(i) \leftarrow \frac{deg(v)+|O_v(i)|}{2*deg(v)+N_v+1}$
12: $p_v(i) \leftarrow \frac{p_v(i)}{\sum_{l \in Ne_v} p_v(l)}$
13: $N_v \leftarrow \sum_{j \in Ne_v} deg_v(j)$
14: $deg_v(i) \leftarrow \arg\max_k \{Pr_v(deg(i) = k | p_v(i), N_v)\}$
15: **end if**
16: **end for**

Definition 5.3.7 is implemented in Algorithm 4 to estimate the degree of each neighbor and the second degree of a node. Each node executes Algorithm 4 locally to estimate degrees of its neighbors without help of global knowledge about the network. From line 4-6 in Algorithm 4, the node initializes the variables as explained in Definition 5.3.7. From line 11-14, v updates estimations of $deg_v(i)$ upon each *observation*. In line 14, in particular, the binomial distribution from Equation (5.1) is used to find the degree which gives the maximum probability of $Pr_v(deg(i) = k | p_v(i), N_v)$, where k is tested from 1 to N_v. In line 12, *Barabási-Albert* algorithm [33] is applied after each estimation to redistribute $p_v(i)$

since many social networks are known to follow a power-law degree distribution [40].

5.3.4 Stopping criterion for the algorithm

Observations can be unlimited for real social networks. Therefore, without using a stopping criterion, a node could run Algorithm 4 forever. We use the velocity of *observations* as a soft stopping criterion. Notice that our algorithm is an online algorithm that can stop and restart depending on the velocity value defined in Definition 5.3.8. It is a soft stopping criterion because the value of velocity changes positively or negatively over time.

Definition 5.3.8 (Velocity) *Consider a node v and its $O_v(i)$. Given any two consecutive observations from $O_v(i)$, say $o_v(i,t')$ and $o_v(i,t)$ where $t' < t$, the velocity associated with $o_v(i,t)$ at time t, i.e., $vel(o_v(i,t))$, is $\frac{1}{t-t'}$. If the observation $o_v(i,t)$ is the first observation in $O_v(i)$ then $vel(o_v(i,t))$ is zero. Notice that time of occurrence, say t, is relative time.*

Upon each *observation*, an observer node (in our algorithm, node v) not only updates its belief about the degree of the *observee* node (in our algorithm, node i), but also compute the velocity associated with each *observation*.

Definition 9 explains how velocities of observations are used to stop and resume the degree estimation process.

Definition 5.3.9 (Stopping criterion) *Given a node v and a neighbor i, for each observation $o_v(i,t)$, v can compute the sum of the velocities from 0 to t using $\int_0^t vel(o_v(i,t))\mathrm{d}t$. Algorithm 4 stops the degree estimation of i if $\int_0^t vel(o_v(i,t))\mathrm{d}t < 0$ and begins the estimation process again when the integral becomes positive, $\int_0^t vel(o_v(i,t))\mathrm{d}t \geq 0$.*

Notice that the stopping criteria only applies to the neighbors of v that satisfy the conditions described. At any given time, a node v can refer to the current estimated degrees of neighbors.

5.3.5 Proportionality

Proportionality is a ratio of *observations* to the degree of each neighbor. We formally define *proportionality constant* in Definition 5.3.10. The *proportionality constant* is used only for analysis purposes since true degrees of neighbors are not known to each node.

Definition 5.3.10 (Proportionality constant) *We define the proportionality constant,* $c_v(i)$*, as* $\frac{|O_v(i)|}{deg(i)}$ *for all* $i \in Ne_v$*.*

Note that $c_v(i)$ equals 1 only when the number of v's observed interactions of i, $|O_v(i)|$, is the same as i's degree, $deg(i)$. Since it is impractical that v observes the exact same number of interactions of i as i's degree, i.e., $c_v(i) = 1$ for all $i \in Ne_v$, merely counting the number of *observations* to estimate neighbors' degrees is not enough.

5.4 Experiments

We present two experiments to evaluate our algorithm. First, we apply the algorithm on a scale-free network created by *Barabási-Albert* Model [33]. Then we apply the algorithm to a real world social network data from *Facebook*.

5.4.1 Degree Estimations on Neighboring Nodes in *Barabási-Albert* Network

We generated a scale-free network based on *Barabási-Albert* model using *Cytoscape* [41]. We consider each edge as an interaction between the two nodes connected; this is the same as the base case when the edge belonging to these two nodes represents the only interaction made by the two nodes. Recall that an *observation* is defined as $o_v(i, t)$ where v is the observing node, i is the observed node, and t is the time of the interaction. Then, we generate the time stamps t using random assignments. For example, consider a node v that has two neighbors, i and j, with degrees 1 and 3 respectively. Then, v's observation set includes

4 *observations*, i.e., $O_v = \{o_v(i,t_1), o_v(j,t_2), o_v(j,t_3), o_v(j,t_4)\}$ where $\{t_1, t_2, t_3, t_4\}$ is a randomly generated ordered time sequence.

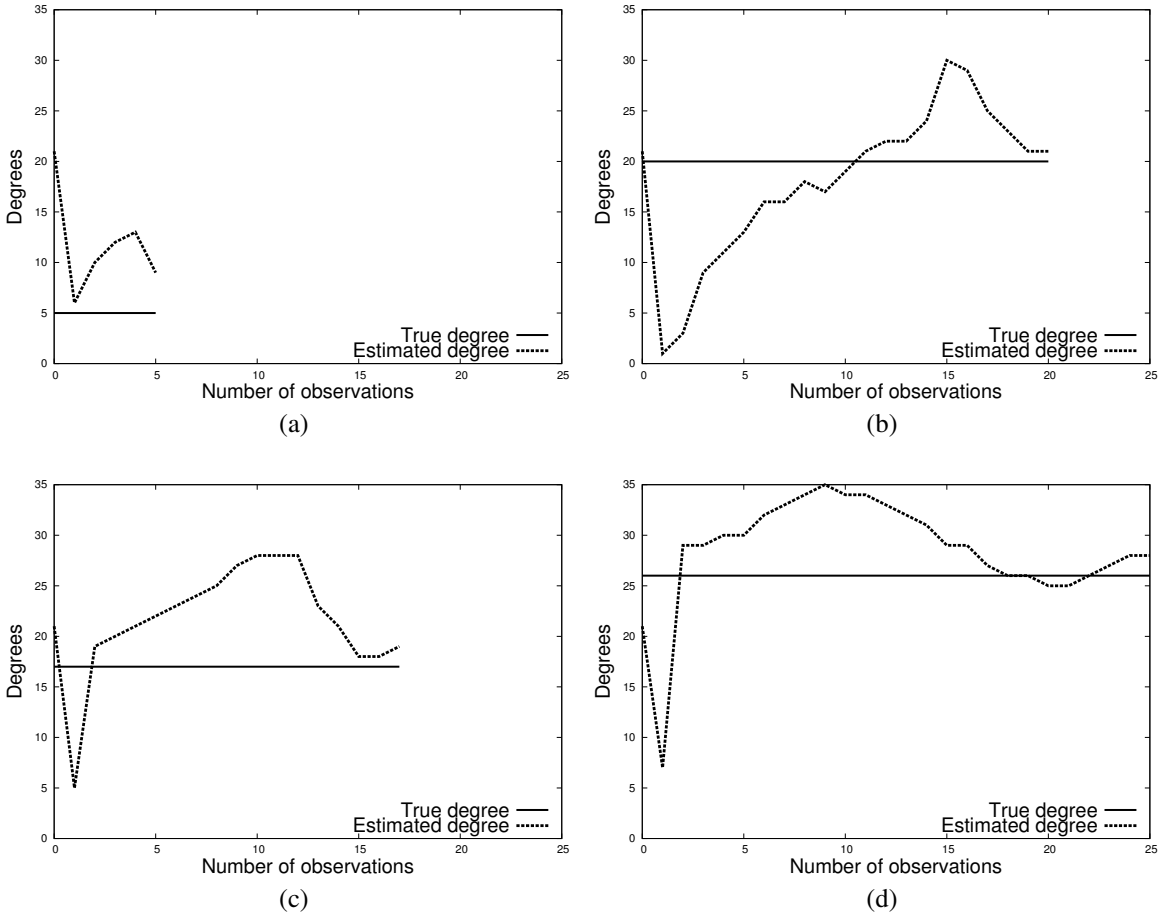


Fig. 5.3: Estimated neighbors' degrees over each *observation*.

The generated scale-free network has 300 nodes and 2561 edges. To study an average behavior, we conducted multiple experiments with different time sequences. The error is measured as the estimated degree divided by the true degree for each neighbor subtracted from 1, i.e., $(1 - \frac{deg_v(i)}{deg(i)})$ for all $i \in Ne_v$ for a node v, where the perfect accuracy is 1. Then we compute the mean squared errors of the accuracies for each node. We define the mean squared error (MSE) for each node, v, as $\frac{1}{deg(v)} \sum_{i \in Ne_v} (1 - \frac{deg_v(i)}{deg(i)})^2$.

Figure 5.3 shows changes of estimated degrees over each *observation*. We picked a node with a typical behavior, which has 21 neighbors and we show the second degree

Noise probability	0	0.2	0.4	0.6	0.8
MSE	0.6111	0.6132	0.6921	0.6831	0.7092
MSE $_{(deg > 5)}$	0.2037	0.2151	0.2265	0.2297	0.2394

Table 5.1: Mean squared errors with increasing noise.

estimations on four of the neighbors. The straight lines in each subfigure represent the true degrees of the observed nodes ($5, 20, 17$ and 26 respectively). The initial estimation of neighbors' degrees is the degree of the observer node, which is 21 in this case. x-axis shows the number of *observations* so far and *y*-axis shows the estimated degrees of the observed nodes over the *observations* made so far. For example, in Figure 5.3a, the observer node had 5 *observations* on the *observee* node and the estimated the degree of the *observee* is 9 at the end of *observations*. In Figure 5.3b and 5.3c, the estimations are reasonably accurate (20 to 21 and 17 to 19 respectively). On the other hand, in Figure 5.3a the estimation is 9 but the true degree is 5. This is because the true degree value is quite small. Our algorithm seems to have a lower bound for the true degree values for a reasonable performance. We show this result in Table 5.1 by comparing mean squared errors for the nodes with degrees above 5.

We show MSEs with different noise probabilities in Table 5.1. We also show the results of nodes with degree values above 5 only. When the noise probability is 0, we use degrees of nodes as number of *observations*, i.e., $c_v(i) = 1$ for all $v \in V$ and $i \in Ne_v$. Then, we add or remove an *observation* from each node with the probability of $0.2, 0.4, 0.6$ and 0.8, respectively for each experiment. As the table shows MSE values do not increase much over increased noise probabilities.

We also computed MSEs for nodes that have degrees greater than 5 only since nodes with very small degrees have relatively high value of MSE. Since each observer node's initial estimations for neighbors' degrees are the degree of itself, it takes a number of *observations* to converge to true degrees of neighbors. However, when the number of *observations* is too small, i.e., small degrees, in this case, the resulting estimation is likely to produce

Proportionality constant (c)	1	2	3	4	5
MSE	0.6111	0.8104	0.8120	0.8047	0.8096
MSE $_{(deg > 5)}$	0.2037	0.2128	0.2001	0.2205	0.2097

Table 5.2: Mean squared errors with different proportionality constants.

more errors. For example, if a node with degree 1 was estimated to have degree 3, than the MSE is 4 which hurts the average MSE. We observe that the algorithm is resilient to noises since MSE values do not increase much as more noise is added.

We test the performance of the algorithm with different proportionality costants, i.e., $\frac{|O_v(i)|}{deg(i)} = c$ for all $v \in V$ and $i \in Ne_v$, where $c = 1, 2, 3, 4, 5$ (Note that we only multiply the number of *observations* since degrees of nodes are fixed). This is to test if the algorithm can tolerate numerous number of *observations* as it often happens in real world applications. Table 5.2 shows MSEs with different proportionality constants defined in Definition 5.3.10. We use the proportionality constants as estimators to compute the MSEs in each experiment. We observe that both MSE and $MSE(deg > 5)$ stays about the same as c increases, which implies that the algorithm can estimate degrees of neighbors with accurate proportionality.

5.4.2 Degree Estimations on Neighboring Nodes in *Facebook* User Network

In real applications of social networks, the number of *observations* rarely agrees with (if not at all) the degrees of observed nodes (*observees*). In *Facebook* user network, some *Facebook* users may not have any activities at all even when they have many *friends* and other users may have more activities than the number of *friends*. Such "activities" include postings on the *walls* (observable) and reading the postings from the *walls* (not observable). If the *observee*'s *page* is actively engaged with other users, the observer can observe more *observations* of the *observee* than its number of *friends*. Also, if the *observee* is not en-

Statistics	Value
♯ Nodes	60867
♯ Edges	1048576
Average degree	17.22
Network diameter	15
Average Path Length	4.39
Modularity	0.602
♯ Communities	219

Table 5.3: Statistics of the *Facebook* user network data.

gaged, the observer may not encounter any *observation* on the *observee*. We examine our algorithm with real *Facebook* user network data to evaluate how the algorithm performs when not all interactions are observed.

Facebook is one of the most popular and widely used online social network all around the world. By the end of March 2014, *Facebook* had 1.28 billion monthly active users worldwide and 802 million users log on to Facebook daily, according to *Facebook* newsroom. We used data sets from [42] which contains links between users and communications (collected from September 14, 2004 to January 22, 2009) among users via wall feature. The resulting network from user links consisted of $60,867$ nodes and $1,048,576$ edges. Some statistics of the network are given in Table 5.3.

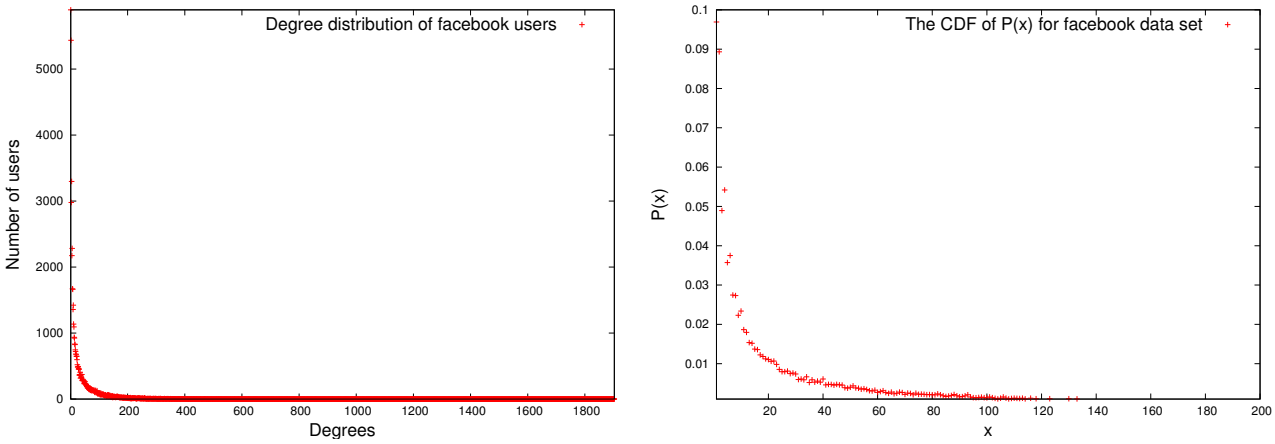
The smallest degree in the network is 1 and the largest degree is $1,903$ where the average degree is approximately 17, which tells us there are only a few users with very large degrees. Also, the maximum distance between two users (network diameter) is 15 and the average distance between any two users is 4.39 which are comparable with other social networks presented in [43]. Finally, the modularity of 0.602 is considered relatively high as [44] presented that the value usually falls between 0.3 and 0.7.

One of the important and innovative assumptions that our algorithm makes is that the degree distribution of the social network follows a power-law distribution. In Figure 5.4 (a), we show that the degree distribution of *Facebook* users from the data obeys power-law. In Figure 5.4 (b), we show the cumulative distribution function of x, $P(x)$, where x is a

	MSE	MSE(>50)	Observed	Estimation Ratio
Average	1.3906	0.7345	26.90%	0.9214

Table 5.4: Statistics of estimation results on *Facebook* user network.

degree of a node. We highlighted the graph presented in Figure 5.4 (b) to clearly see the power-law distribution (x ranges from 1 to 200). To verify if the degrees of *Facebook* users truly follow a power-law distribution, we use a goodness-of-fit test via a bootstrapping procedure, suggested by [13]. The hypothesis, that the data set actually follows a power-law, is tested by estimating the model parameters and then calculating the goodness-of-fit between the data and the power-law. We used *poweRlaw* package by [45] to perform the test. According to [13], the hypothesis, that the given data points follow a power-law, is valid if the resulting value is greater than 0.1. *Facebook* user data set produced a value of 0.98 which proves that the degree distribution of *Facebook* users follows a power-law.



(a) (b)

Fig. 5.4: Degree distribution of *Facebook* users follows power-law.

The *Facebook wall* communication data shows the source (the user who writes to a wall), the target (the user whose wall has received a message from the source), and the time of the interaction (when the message was written to the wall). Notice that reading activities are not included in the data. Any user in *Facebook* can run our algorithm to find out how many *friends* each of its *friend* has.

To test the performance of our algorithm, we run this algorithm for each user, for each *observation* (wall communication) from its neighbors. Table 5.4 shows the results of how the algorithm performs given *Facebook wall* activities. We compute the deviations from the perfect proportionality (Estimation ratio) assuming the proportionality constant is 1, i.e., $1 - \frac{|O_v(i)|}{deg(i)}$. In the table, *Observed* column shows the average percentage of *observations* each user could make. For example, if the degree of a user a is 10 and its neighbor b could observe 10 wall communications of a, than it is 100% *observed*. In this dataset, only 26.9% is observed which is challenging for the algorithm. It is comparable with the case when the proportionality constant (c) is 4 in Table 5.2 (since 25% observed is approximately when $c = \frac{1}{4}$). As *MSE* and *MSE(>50)* columns in Table 5.4 shows, the results are not as good, compared with the synthetic scale-free network used in Section 5.4.1. However, *MSE(>50)* of the estimation is 0.7345 which is close to the results from the generated network. Also, the average *Estimation Ratio* (estimated degree divided by true degree) is 0.9214 (1 being the perfect estimation) which is high.

If we purely count the number of *observations* to estimate degrees of neighbors, the *Estimation ratio* would be only 0.2690 compared to 0.9214. By applying our algorithm to estimate degree of neighbors, we achieved 0.9214 accuracy ratio which is more than three times better result.

5.5 Chapter Summary

Traditional research focused on estimating degree distributions using macro-level algorithms. Because the size of network is huge, usually sampling is made to estimate degree distributions [38]. Our focus is to compute a precise degree of each neighboring node from an observer node within the network (therefore, micro-level approach). Our algorithm accommodates dynamic natures of online social networks, introducing the notion of *observations* which are obtained from interactions (or communications) among nodes

(users). However, if we merely count the number of *observations* for the estimated degree of a neighbor, only active neighbors are discovered. In other words, in *Facebook* wall communications, if we simply count the number of *observations* to be estimated degrees of observed nodes, only the users who explicitly write on *observee*'s walls are counted. In reality, there are more readers (who silently read communications of others) than writers (who writes on walls). Although readers are currently inactive, they are potential writers and they should be also considered when estimating degrees of neighbors. The proposed algorithm combines the concept of Bernoulli trials and a power-law distribution to reason about hidden neighbors' of neighbors (readers).

In our experiments, we tested the algorithm on a synthetic scale-free network and *Facebook* user network. For the scale-free network experiment, we presented the mean squared errors of the accuracies. We added noise *observations* to show that the algorithm can estimate degrees of neighbors with incomplete information. In *Facebook* user network experiments, we tested the algorithm with wall communications among users as *observations*. Due to incomplete *observations*, average proportionality constant is as low as 0.2690, the results are not better than that of the scale-free network experiment.

The proposed algorithm is based on the assumption that the number of activities of nodes is positively related to the degrees of nodes. In real life, the assumption is reasonable because people who have more connections have more social activities compared to people who do not.

However, there are certain relationships that are unique to online social networks. For example, many celebrities are neighbors with fans on online social networks. In this case, it becomes difficult to estimate the degrees of the celebrities from their neighbors point of view because they only interact with very few of their online neighbors. Note that, if degrees are estimated based only on the number of interactions (*observations*), hidden neighbors (connected but never interact with) may not be discovered. Our algorithm can capture hidden but potentially active neighbors because it can infer about unseen activities

through the mechanisms of Bernoulli trials and power-law distributions.

Our algorithm can be further improved by utilizing additional information. In some applications, types of activities matter and certain types of *observations* are more valuable in estimating degrees of neighbors. By selectively using *observations* instead of all the *observations*, we can improve quality of the estimations. For example, on *Twitter* network, activities such as *follow* should be weighted more compared to the activities such as *tweet* and *retweet*. We can also extend our work by applying our algorithm recursively to enable users to capture the global view of the network, e.g. degree distributions of the entire network it belongs.

CHAPTER 6
CONVERGENCE OF TRUE COOPERATIONS IN BAYESIAN REPUTATION GAME

In this Chapter, we introduce Bayesian games where players have asymmetric information about each other. In Bayesian games, players have initial beliefs about the type of other players and updates the beliefs according to Bayes' Rule. We consider repeated games since the reputation values are used as part of the payoffs, which is often ignored in one-shot games. By introducing reputation in the game, we show how reputation values can influence behaviors (decision making) of agents (players).

6.1 Bayesian Reputation Game

Reputation management is used in many distributed environments to aid decisions of individual entities on who to interact with for their maximal gain. There are two general approaches to reputation management: centralized and distributed. A centralized method require a shared information server that provides a single view of reputations on the enti-

ties. In other words, a reputation of an entity is implicitly agreed upon everyone else in the system. In a distributed management, similar approaches exist. We, however, consider systems in which each entity keeps private reputation values for other entities based on past interactions. For example, in a distributed social network, when a user encounters a new user, it can ask its neighbors (e.g., friends) about the new user's reputation. The user can ask multiple neighbors and average the responses or among its neighbors, the user selects a neighbor who are most likely to answer truthfully based on the neighbor's reputation and the degree. We propose a *bayesian reputation game* that captures the motivation of users to accept the request and answering truthfully. Our model is unique in that unlike traditional game theory models, there is no immediate reward structure for actions of players. It is however possible to analyze the strategic interactions using game theoretical analysis as discussed later.

In Section 4.2.7, we introduced *ask function* which agents can use to inquire about other agents reputation values. When an agent wants to know reputation of other agents, it asks to all of its neighbors and aggregates the answers proportionally weighted by degrees of neighbors. However, there exist some disadvantages for asking to all of the neighbors at the same time compared to selecting a neighbor to ask. First, asking to everyone around (all the neighbors) is computationally more expensive than asking to a selected neighbor (or a few of selected neighbors). For example, if an agent with 100 neighbors sends request to all of its neighbors, it needs to compute weights for all the neighbors to aggregate all the answers and also it has to wait for all the neighbors to respond. Second, neighbors are more likely to respond quickly, if at all, if they are asked for a favor individually. In other words, individuals feel more responsibility when they take charge of an entire task. Third, it is hard to detect deceptions when neighbors intentionally submit incorrect answers and the agent is better off by selecting a neighbor who is the most likely to give a truthful answer.

Therefore, agents are encouraged to be selective about whom they ask. In the following sections, we describe how agents can select suitable neighbors to ask and the resulting

consequences in game theoretic formulation.

6.2 Previous Studies on Reputation Computation

6.2.1 Trust and reputation management

The concept of trust and reputation is employed by many disciplines including Internet/mobile security, E-commerce, multi-agent systems and social networks. Many methods have been proposed in the literature [17, 19, 46–49] for reputation management.

Reputation captures behavioral history of the counterpart while trust is used to predict expected future behaviors. Therefore, being able to derive trust from reputation is important for decision makings under uncertainties. For example, in online commerce systems, buyers rely on reputation of sellers to select whom they want to interact with and, therefor, sellers want to increase their reputation values to attract more buyers. Our proposed game suggests that we can achieve convergence of true cooperation by using reputation as expected return in interactions.

Reputation contains two types of information. One comes from direct interactions with immediate neighbors of a node and the other comes from aggregating information from indirect neighbors. Since the evaluation of direct interactions is application dependent, many studies focus on designing algorithms for aggregating reputation. [17] and [47] discuss reputation aggregation mechanism when a node receives information from its neighbors. [17] introduces a distributed voting mechanism where the weights of votes are proportional to the degrees of voters. [47] introduces a differential gossip algorithm where the gossip weights depend on trust values of nodes. Our algorithm is also motivated by the process of aggregating reputation. In our game, although there is no direct reward for voting or replying when asked about reputation of others, players are encouraged to cooperate by the expected increase in their reputation held by the requesters.

6.2.2 Bayesian games for online reputation

Online portals or communities employ reputation mechanisms to discourage non-cooperative behaviors or free ridings. When the mechanism is properly designed, the users are encouraged to cooperate by punishing the misbehaving ones with an implicit threat of lowering reputation values. To be able to utilize reputation as incentives, the users should be interested in future interactions. In game theoretic terms, we consider repeated games since reputation is simply ignored in one time games for it has no effect to future outcomes.

Many studies have been presented to explain real world scenarios. [50] introduces a game-theoretic model for reputation in online auctions, especially for eBay auction site. They define reputation games where each stage game is bayesian game with asymmetric information, i.e., the buyer does not know whether the seller is honest or not. [51] assumes that uniform punishment strategies are available and proposes that the Nash Equilibria, under incomplete information, of infinitely repeated game are equivalent to payoffs of the equilibria where players reveal complete information. [52] presents a collective reputation model and studies under what environments the reputation is persistent. They state that the reputation of a group can persist if the environment evolves stochastically as actions of group members are strategic complements.

Unlike other games where payoffs are physical (or monetary) gains, we utilize private reputation values as expected payoffs. In our game, although the actions taken by both players are disclosed after the game, the payoff for each player is not shared. Therefore, the game models incentives of players to be cooperative when there is no exchange of goods.

6.3 Bayesian Games

In this section, we introduce *Bayesian reputation game*. A *Bayesian reputation game* is a non-zero sum game where players can compute payoffs for each action based on reputation

and estimated degrees of the opponents. Figure 6.1 shows two neighboring agents, i and j, having reputation and estimated degree values for each other. We consider nodes in a network are players and each node can inquire about reputation of another node.

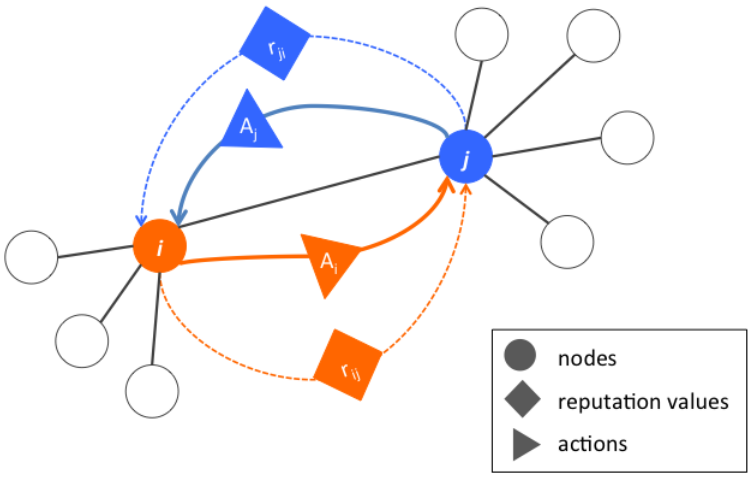


Fig. 6.1: Players i and j on a network.

6.3.1 Definitions of *Bayesian reputation game*

A *Bayesian reputation game* is a tuple (N, A, Θ, p, u) where:

- N is a set of agents;
- A^{row}= *{Ask, Not Ask}*, A^{column} = $\{C_{True}, C_{False}, D\}$, where A^{row} is the set of actions available to the row players and A^{column} is the set of actions available to the column players;
- Θ^{column} = *Honest* $\times$ *Dishonest* where Θ_j is the type of player j;
- $p_{ij} \mapsto [0, 1]$ is player i's estimated probability of player j being type *Honest*; and
- Utilities u_i: $A^{row} \times A^{column} \times \Theta \rightarrow \mathbb{R}$

A player who initiates the game is the row player and the column player follows. Row players and column players have different sets of actions available, i.e., A^{row} and A^{column}.

Column players can be either type *Honest* or *Dishonest*. The type of a column player is only known to itself and the row player guesses the type the column player based on the degree and the reputation of the column player.

Figure 6.2 explains the scenario of the game sequentially. The row player i decides whether to take *Not Ask (NA)* or *Ask (A)*. If i chooses *NA* the game is over and both players i and j gets payoffs of zero. The game gets more interesting when i takes *A*. If j is of type *Honest*, j can cooperate to i by giving truthful information (C_T) or defect by denying to answer (*D*). If j is *Honest* and it has no information on what was inquired by i, j should defect (D). On the other hand, if j is of type *Dishonest*, either j tries to deceive i by intentionally providing false answer (C_F) or it can decide not to answer (*D*). The payoffs for both players are given at the end of the tree. We represent the same game in normal form in Table 6.3.

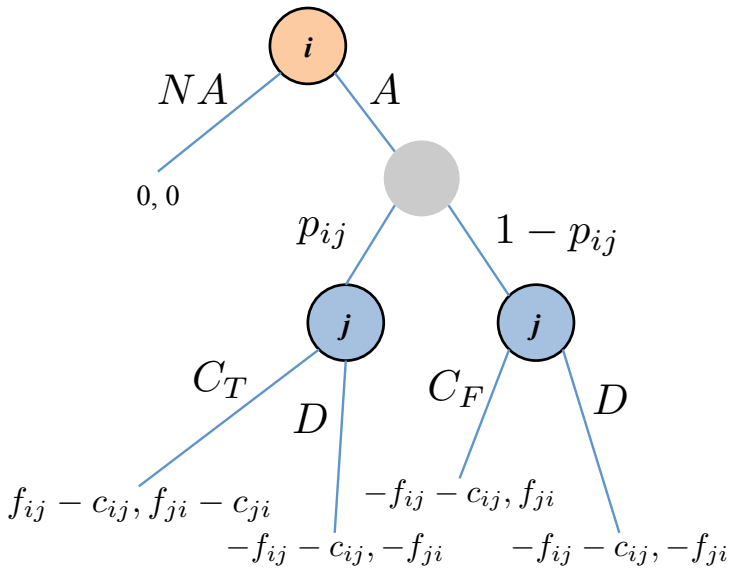


Fig. 6.2: Single stage reputation game.

Before we discuss the payoffs in Table 6.3, we define f_{ij} as follows.

$$f_{ij} = \frac{(r_{ij} + 1)}{2} \times \frac{d_{ij}}{max_{d_i}} \tag{6.1}$$

	C_{True}	D
Ask j	$f_{ij} - c_{ij}^{ask}, f_{ji} - c_{ji}^{answer}$	$-f_{ij} - c_{ij}^{ask}, -f_{ji}$
Not Ask	$0, 0$	$0, 0$

Table 6.1: Type *Honest*

	C_{False}	D
Ask j	$-f_{ij} - c_{ij}^{ask}, f_{ji}$	$-f_{ij} - c_{ij}^{ask}, -f_{ji}$
Not Ask	$0, 0$	$0, 0$

Table 6.2: Type *Dishonest*

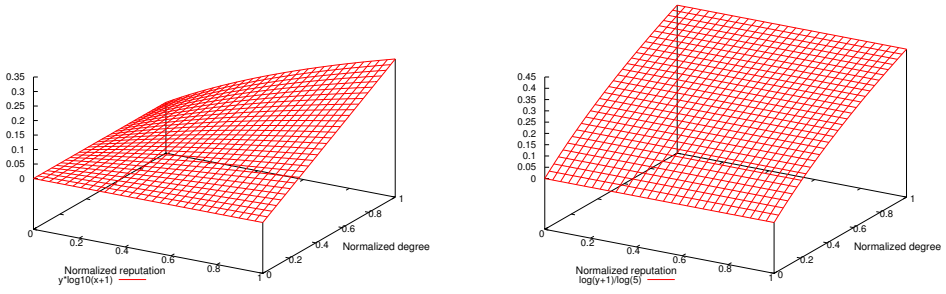
Table 6.3: *Bayesian reputation game* in normal form.

f_{ij} represents i's expected gain based on j's reputation and the degree. Therefore, when i takes *Ask* as its action and j takes C_{True} from Table 6.1, i's payoff is $f_{ij} - c_{ij}^{ask}$ where c_{ij}^{ask} is i's cost of taking *Ask*. Likewise, j's payoff is $f_{ji} - c_{ji}^{answer}$ where c_{ji}^{answer} is j's cost of computation (i.e., the cost of computing the answer). In Table 6.2, where j is of type *Dishonest*, the payoff for i when j takes C_{False} is $-f_{ij} - c_{ij}^{ask}$ because, despite i's cost to ask, j deceives i by giving a non-truthful answer. However, j's payoff is f_{ji} since there is no occurred cost for j and j thinks that i does not know the answer is false and appreciates it. When i decides not to ask (*Not Ask*), the payoffs for both player is 0.

Here we define the cost function of asking which is a logarithmic function dependent on the reputation and the degree of the opponent.

$$c_{ij}^{ask} = \frac{d_{ij}}{max_{d_i}} \times \log_{10}(\frac{r_{ij} + 1}{2} + 1) \tag{6.2}$$

The cost function for asking, c^{ask}, is similar to the reward function, defined in equation (6.1), in that it considers the reputation and the degree of the opponent. When i wants to inquire j about some information, if j is reputable (high reputation) and influential (high degree), i's expected satisfaction (f_{ij}) is high if j cooperates and i's disappointment is high ($-f_{ij}$) if j defects. The occurred cost (c_{ij}^{ask}) reflects approachability as it is harder to approach an individual if he is reputable and influential.



(a) (b)

Fig. 6.3: Cost functions for *ask* and *answer*.

Likewise, we define the cost function for answering which is a logarithmic function dependent on its own degree.

$$c_{ji}^{answer} = \log(\frac{d_j}{max_{d_j}} + 1) \tag{6.3}$$

The cost of answering is only dependent on its degree since when j is asked, j gathers information from its neighbors. It is worth noting that both functions are bounded by 0 and 1 ($0 \leq f, c \leq 1$).

Figure 6.3 shows the cost functions (6.2) and (6.3). As seen in Figure 6.3a, the value of c_{ij}^{ask} increases as reputation and degree increases while the value of c_{ji}^{answer} only depends on degree as shown in Figure 6.3b.

In Figure 6.4, we show how a row player i estimates the type of the column player j, p_{ij}, based on estimated reputation and the degree of j. Player i assumes that j is more likely to be of type *Honest* as j's reputation values and j's relative degree (compared to other neighbors of i) is higher. More specifically, if a point $(r_{ij}, \frac{d_{ij}}{max_{d_i}})$ is on the line $r_i + 2\frac{d_i}{max_{d_i}} - 1 = 0$, p_{ij} is 0.5. If $(r_{ij}, \frac{d_{ij}}{max_{d_i}})$ is above the line, p_{ij} is 0.5 plus the point-line distance and vice versa.

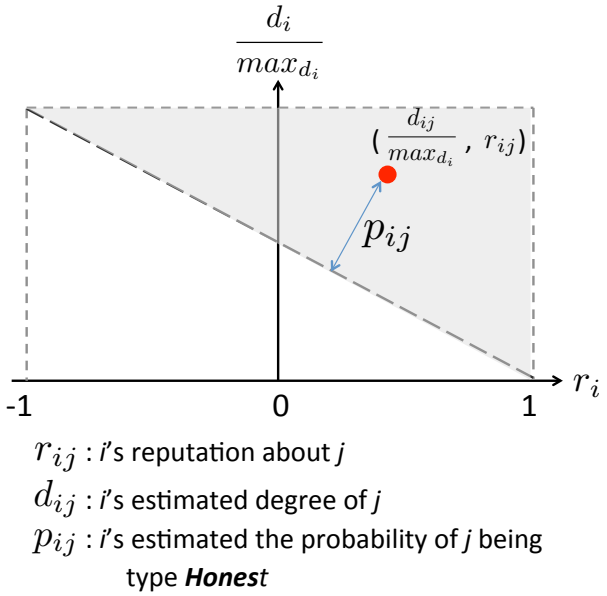


Fig. 6.4: Type estimation based on reputation and degrees

	$C_{True}C_{False}$	$C_{True}D$	DC_{False}	DD
Ask j	$(2p_{ij}-1)\cdot f_{ij} - c_{ij}^{ask}$, $f_{ji} - p_{ij}\cdot c_{ji}^{answer}$	$(2p_{ij}-1)\cdot f_{ij} - c_{ij}^{ask}$, $(2p_{ij}-1)\cdot f_{ji} - p_{ij}\cdot c_{ji}^{answer}$	$-f_{ij} - c_{ij}^{ask}$, $(1-2p_{ij})\cdot f_{ji}$	$-f_{ij} - c_{ij}^{ask}$, $-f_{ji}$
Not Ask	$0, 0$	$0, 0$	$0, 0$	$0, 0$

Table 6.4: Extended form of *Bayesian reputation game*.

We formally define p_{ij} in (6.4). As referenced from *ReMSA* [17], we assume $-1 \leq r_{ij} \leq 1$.

$$p_{ij} = \begin{cases} \frac{1}{2}(1 + \frac{|1\cdot r_{ij} + 2\cdot \frac{d_{ij}}{max_{d_i}} - 1|}{\sqrt{1^2+2^2}} \cdot \frac{\sqrt{1^2+2^2}}{2}) & \text{if } \frac{d_{ij}}{max_{d_i}} \geq -\frac{1}{2}r_{ij} + \frac{1}{2} \\ \frac{1}{2}(1 - \frac{|1\cdot r_{ij} + 2\cdot \frac{d_{ij}}{max_{d_i}} - 1|}{\sqrt{1^2+2^2}} \cdot \frac{\sqrt{1^2+2^2}}{2}) & \text{if } \frac{d_{ij}}{max_{d_i}} < -\frac{1}{2}r_{ij} + \frac{1}{2} \end{cases} \tag{6.4}$$

6.3.2 Computing Bayesian Nash Equilibrium (BNE)

Pure Strategy BNE

We collapse incomplete information problem as a static extended game with all possible strategies, assuming the probability of a column player being the type *Honest* (Table 6.1) is p_{ij}. The new extended game is shown in Table 6.4.

For a row player, from Table 6.4, there is no dominant strategy since the payoff of *Ask*

		y	$(1-y)$
		C_{True}	D
x	Ask j	$f_{ij} - c_{ij}^{ask}, f_{ji} - c_{ji}^{answer}$	$-f_{ij} - c_{ij}^{ask}, -f_{ji}$
$(1-x)$	Not Ask	$0, 0$	$0, 0$

		z	$(1-z)$
		C_{False}	D
x	Ask j	$-f_{ij} - c_{ij}^{ask}, f_{ji}$	$-f_{ij} - c_{ij}^{ask}, -f_{ji}$
$(1-x)$	Not Ask	$0, 0$	$0, 0$

Table 6.5: Mixed strategy

j could be positive or negative depending on f_{ij} and c_{ij}^{ask} values.

For a column player,

- when $2 \times f_{ji} = c_{ji}^{answer}$, then the payoffs for the strategies $C_{True}C_{False}$ and DC_{False} are equal and $C_{True}D$ and DD are also equal. However, $C_{True}C_{False}$ and DC_{False} always dominates $C_{True}D$ and DD;
- when $2 \times f_{ji} > c_{ji}^{answer}$, then $C_{True}C_{False}$ is the dominant strategy;
- when $2 \times f_{ji} < c_{ji}^{answer}$, then DC_{False} is the dominant strategy.

Mixed Strategy BNE

In order to obtain mixed strategies, assume the probabilities of playing each action are as follows. A row player plays *Ask j* with the probability of x, the column player of type *Honest* plays C_{True} with the probability y and the column player of type *Dishonest* plays C_{False} with the probability of z. The assignment of the probabilities are shown in Table 6.5.

The players best responses are as follows.

- The row player would play *Ask j* instead of *Not Ask* if

  $p[y(f_{ij}-c_{ij}^{ask})+(1-y)(-f_{ij}-c_{ij}^{ask})]+(1-p)[z(-f_{ij}-c_{ij}^{ask})+(1-z)(-f_{ij}-c_{ij}^{ask})] > p[y0+(1-y)0]+(1-p)[z0+(1-z)0]$.

  $(i)\ 2ypf_{ij} - f_{ij} - c_{ij}^{ask} > 0 \Rightarrow x = 1.$

- The column player's best response with type *Honest* (Table 6.1): the column player would play C_{True} instead of D if $x(f_{ji} - c_{ji}^{answer}) + (1-x)0 > x(-f_{ji}) + (1-x)0$.
  (ii) $x > 0, 2f_{ij} > c_{ji}^{answer} \Rightarrow y = 1$.

- The column player's best response with type *Dishonest* (Table 6.2): the column player would play C_{False} instead of D if $xf_{ji} + (1-x)0 > x(-f_{ji}) + (1-x)0$.
  (iii) $x > 0 \Rightarrow z = 1$.
  (iv) $x = 0 \Rightarrow z \in [0, 1]$. Since we consider the game to be sequential, if $x = 0$, the game ends.

6.3.3 Extensions to repeated games

In the previous section, we have introduced *Bayesian reputation game* and discussed BNE concepts based on a single stage game. We extend the game to be repeated in order to observe behavioral changes of players. In repeated games, players consider not only the payoffs of the current game, but they also consider expected future returns, i.e., how the current action influences future outcomes.

Updating Rewards into Reputation

Each time the game is played, both the row and column players get payoffs described in Table 6.3. The payoffs represent perceived gain of reputation and appreciation. Row players (requesters) update the private reputation values of the opponents after each game played according to given payoffs.

We expect that if the game is played repeatedly, the game promotes C_{True} and the average reputation values for all the players increase.

Selecting a column player

We extend the game in Table 6.3 to a multi-player game. The row player i can select a column player based on utilities of the game. For example, if i has 5 neighbors as in Figure 6.1, i computes expected return from each of its neighbor based on the payoffs given in Table 6.3. Then i plays the game with the selected neighbor, j. This is exactly same as i having $d_i + 1$ possible actions, where d_i is degree of i, which are {*Ask 1, Ask 2, ..., Ask j, ..., Ask* d_i*, Not Ask*}, since i will plays an action which gives the highest expected return, in this case *Ask j*. This process is done by the function $pickCPlayer()$ in Algorithm 4.

Repeated Games with Discounting

In this section, we introduce a repeated reputation game, where games in the future are discounted by a discount factor, $\delta \in [0, 1)$. If players care enough about future returns, then any equilibrium that gives payoffs higher than the stage game NE can be sustained by the threat of playing the stage game NE. We discuss the minimum δ value to sustain C_{True} over C_{False}.

The repeated game $\Gamma(n, \delta)$ is a game where the stage game Γ is repeated n times, and the payoffs are given as below.

$$u(\Gamma(n, \delta)) = \Sigma_{t=1}^{n} u_t(\Gamma)\delta^t \tag{6.5}$$

We want to compare the payoffs of a repeated game for a column player when it plays C_{True} and C_{False}. If the column player plays C_{True} in the first stage game, the row player will keep choosing *Ask j* since the column player was selected by the row player which implies that the row player's perception of the column player, i.e., f_{ij}, was high. Hence, the column player can either play C_{True} infinitely or play C_{False} n times and not asked by the row player again, which gives payoffs of 0 from then on.

- If the column player plays C_{True} infinitely, the discounted payoff is

$$u_j(\Gamma(\infty,\delta), C^{\infty}_{True}) = \frac{f_{ji} - c^{answer}_{ji}}{1-\delta}.$$

- If the column player plays C_{False} infinitely, the discounted payoff is

  $$u_j(\Gamma(\infty,\delta), C^{\infty}_{False}) = \frac{f_{ji}(1-\delta^n)}{1-\delta} + 0,$$

  where after playing C_{False} n times, the row player decides not to ask again.

- If the column player plays *Defect* infinitely, the discounted payoff is

  $$u_j(\Gamma(\infty,\delta), Defect) = \frac{-f_{ji}(1-\delta^n)}{1-\delta} + 0,$$

  where after playing *Defect* n times, the row player decides not to ask again.

Therefore, the column player will sustain C_{True} when the game is infinitely repeated, if $\frac{f_{ji} - c^{answer}_{ji}}{1-\delta} > \frac{-f_{ji}(1-\delta^n)}{1-\delta}$ when type is fixed, or $\frac{f_{ji} - c^{answer}_{ji}}{1-\delta} > \frac{f_{ji}(1-\delta^n)}{1-\delta}$ when type is dynamic.

6.4 Simulation of the Reputation Game

In this section, we design a simulation to observe behaviors of the players when the game is played repeatedly. We assume that the players care about the future returns (i.e., they know that the game is repeated) and players are aware of the evaluating nature (i.e., their actions are evaluated by the opponent although they are not informed of the result of evaluation).

6.4.1 The algorithm

We describe how we implemented the *bayesian reputation game* in Algorithm 4. We use a randomly generated scale-free network which has 500 nodes and 5000 edges. Nodes are players and each player can play games with its neighbors. Any player can be a row player or a column player. Each player initializes the reputation values of neighbors to 0, i.e., neutral, and the types for players are assigned ($0 \sim 249$ players are *Honest* and

$250 \sim 499$ players are *Dishonest*). Then each player selects from its neighbors who to *Ask* (*pickCPlayer()*) based on expected payoffs. Then the row player computes which action to take (*computeRACtion()*) and the column player also computes its own action considering its type and discounted expected utility (*computeCAction()*). If the row player played *Ask* (i.e. there was an interaction), the row player updates its private reputation value about the column player based on which action the column player played.

Algorithm 4 Simulation of infinitely repeated bayesian reputation game

Input: $P =< p_i >$
Output: $A =< a_i >$
for all $p_i \in P$ **do**
  for all $p_j \in Ne_i$ **do**
    $r_{p_i p_j} \leftarrow 0$
  end for
end for
for all $p_i \in P$ **do**
  $rPlayer \leftarrow p_i$
  $cPlayer \leftarrow p_i.pickCPlayer()$
  $rAction \leftarrow rPlayer.computeRAction(cPlayer)$
  $cAction \leftarrow cPlayer.computeCAction(rPlayer)$
  if $rAction = Ask$ **then**
    $rPlyer.updateReputation(cPlyer, cAction)$
  end if
end for

6.4.2 Results

In this section, we show empirical results of the proposed reputation game played according to Algorithm 4. We use the scale-free network for the following experiments as most peer to peer and social networks follow the power law distribution [53]. The network has 500 nodes (players) and 5000 edges (relations).

Convergence of actions

Figure 6.5 shows proportions of each action taken by a column player when a row player takes *Ask*. We omit the action D_{False} since it is dominated by C_{False} and never played. Figure

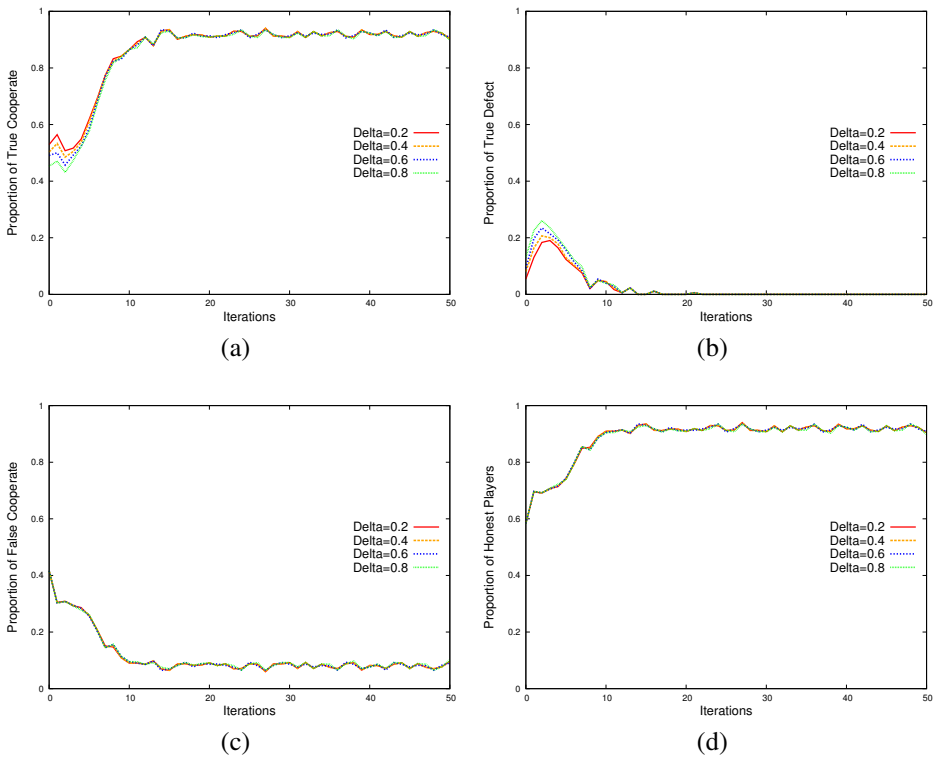


Fig. 6.5: Proportions of actions with different δ values and proportion of the number of *Honest* players.

6.5a shows that the true cooperation (C_{True}) is sustained after the game is repeated 15 times. Figure 6.5b shows that D_{True} dies out after around 20 iterations. Figure 6.5c shows that C_{False} decreases as the game is repeated and converges to 0.1 (played at 10% of the time). Figure 6.5d shows the proportion of the *Honest* players (when the row players decide to *Ask*) over time. The proportion of the *Honest* players quickly converges to a very high rate, near 0.9. Since, in this experiment, players cannot change their types, the increase in the proportion of *Honest* players implies that the *Dishonest* players are less asked by row players.

Convergence of reputation

In the previous section, we showed that the choice of action converges to true cooperation. Since row players will choose *Not Ask* when expected payoffs from all the neighbors are

negative (the payoff for *Not Ask* is zero), the rate of taking *Ask* decrease over iterations. However, in some applications such as auctions, sometimes, it is required to take the best offer (or bid). Therefore, we show how reputation values of players change in both cases, where row players are allowed to take *Not Ask* and where row players are required to take *Ask* to the best available neighbor. The value n is set to 2 unless mentioned otherwise.

Figure 6.6 shows the convergence of overall reputation values of the players in each scenario. Although Algorithm 4 is distributed and reputation values are private, for the purpose of discussion, we summed up perceived reputation values for each player and divided by the number of neighbors. For example, in Figure 6.1, the sum of i's perceived reputation is $r_{ji} + r_{ki} + r_{li} + r_{mi} + r_{ni} = \sum_{j \in Ne_i} r_{ji}$, where Ne_i is a set of i's neighbors. Then the representative reputation value of i is $\frac{\sum_{j \in Ne_i} r_{ji}}{|Ne_i|}$. Therefore, the representative reputation value of a player is the average reputation value from its neighbors. Note that we compute this value only for the purpose of analysis. The players are not aware of reputation values of themselves because the values are kept privately by its neighbors.

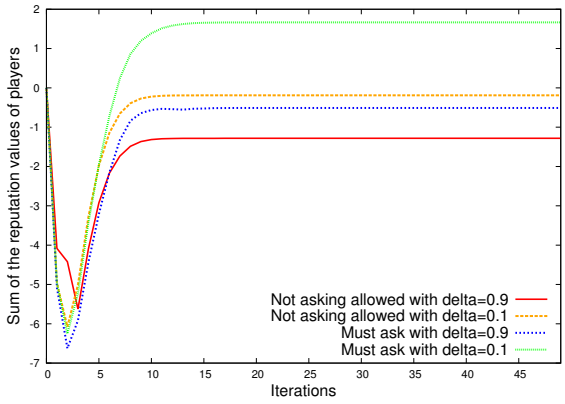


Fig. 6.6: Convergence of the sum of reputation values of players.

As shown in Figure 6.6, the sum of representative reputation values of all players converge after a few iterations, around 10. The plotted lines from all scenarios show very similar behaviors. The global sum starts from zero because the reputation values of all players are initialized to zero, it goes down for a while and then it starts to increase and converges.

The convergence value is higher when row players are required to take *Ask* (must ask) because row players take chances by taking *Ask* to a neighbor with the highest expected return. By exploring more neighbors (although not voluntary), row players have more opportunities to find the best neighbor.

The initial downfalls that are observed in all cases in Figure 6.6 is the periods when row players are exploring their neighbors. In other words, since the initial (private) reputation values of all players are zero, the deviation of expected returns from neighbors is small and therefore row players can explore neighbors for a few initial rounds because the best neighbor (whose expected return is the highest) keeps changing. After a few interactions (iterations), the deviation becomes big enough (there exist a neighbor who achieved high reputation) and the row players exploits the best neighbor from then on.

Dynamic types and convergence behaviors

In the previous experiments, we showed that the actions taken by column players converge to the true cooperation and the sum of overall reputation values of the players increase and converge after a few iterations. However, the types of players were fixed throughout the entire iterations, and therefore, the convergence of actions implies that the row players are more likely to choose the honest column players to get true cooperation reactions. The dishonest players still exist (because the type is fixed) but they fell behind and are excluded from interactions.

To encourage true cooperation even from dishonest players, we design a new experiment where players are allowed to change their types if by doing so they have higher expected payoffs. Every time a column player is picked by a row player, the column player can decide its type based on payoffs from both matrices in Table 6.3.

The results presented in Figure 6.7 shows that dishonest players will eventually change to honest types under appropriate δ and n. The common behaviors observed in all subfigures in Figure 6.7 are: 1) the number of row players who take *Ask* starts from 500 and then

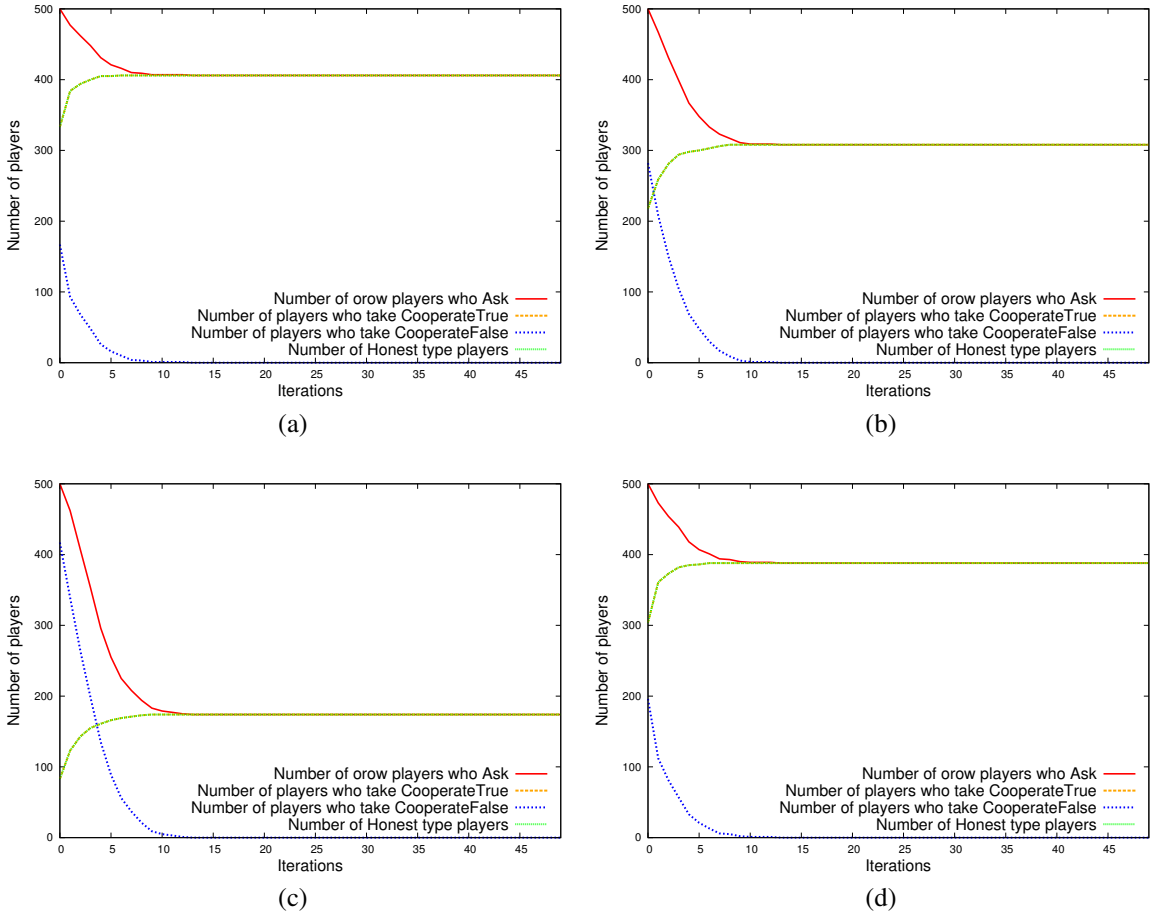


Fig. 6.7: Number of players with different δ and n values.

decreases until it converges. The convergence point meets with the number of players who are type honest, 2) the number of column players with type dishonest (same as the number of players who take C_{False} as the action *Defect* is dominated in dishonest type) decreases over time and dies out, 3) the number of honest type players increases and all the column players who are asked play C_{True}.

Higher δ and lower n values lead to higher convergence point for the number of honest type players (or the number of players who play C_{True}) because higher δ value implies more incentives to player to play honestly for larger future return as well as lower n value. Also, we found out that the similar convergence behaviors are observed for δ value approximately higher than 4.9. As shown in Figure 6.7c and 6.7d, for the same n value, the convergence point is higher for the number of honest type players if δ is higher. *Defect* is not played in this experimental setting because players can now change their types to be dishonest and play C_{False} rather than stay honest and play *Defect* which is dominated by C_{False}.

According to the analysis of the repeated *bayesian reputation game* from Section 6.3.3, when we allow dynamic type changes, the true cooperation is sustained if $\frac{f_{ji} - c_{ji}^{answer}}{1-\delta} > \frac{f_{ji}(1-\delta^n)}{1-\delta}$. Since $1-\delta > 0$, the inequality is the same as $[f_{ji} - c_{ji}^{answer} > f_{ji}(1-\delta^n)] = [c_{ji}^{answer} < f\delta^n]$. Theoretically, the minimum δ value for player j is $\frac{c_{ji}^{answer}}{f_{ji}}$ but since we used the same δ value for all the players, the private δ value is not achieved. We found out that when $\delta > 0.48$, the true cooperation could emerge, similar to the behaviors depicted in Figure 6.7. The maximum n is not bounded which implies that, with high enough δ value ($\delta \rightarrow 1$), players can tolerate bad behaviors (C_{False} and *Defect*) longer and still reach the convergence of true cooperations.

6.5 Chapter Summary

We introduced a *bayesian reputation game* where private reputation values are used as part of payoffs. We assume that all the players want to increase their reputation values because higher reputation gives more opportunities for interactions where players get paid off. Each player becomes a row player and a column player. A row player can make a request (*Ask*) to a column player who is most likely to perform the requested task properly. The process of deciding who to request is based on the discounted expected utility analysis with estimation of types. To test the algorithm, we designed three sets of experiments. First, we follow the proposed algorithm and show the convergence point of each action and the proportion of the action C_{True} is sustained. Second, we enforce row players to request each iteration (i.e., they are not allowed to take *Not Ask*) and compare the global average reputation values of players with the original scenario. Although both scenarios show similar behaviors where the overall reputation values increase and converge, by enforcing the row players to play *Ask* we achieve higher convergence values. Third, to see if players choose to be honest when changing type is allowed, we designed a dynamic type assignment whenever a column player is requested by a row player. The results show that with high enough δ values, the dishonest type dies out and the players converges to the honest type.

We made three assumptions in this paper. One is that all the players are interested in increasing their reputation values held by neighbors so that they have more interaction opportunities. This assumption is reasonable as in game theory the players are assumed to be rational. Second is that all the players care about the future returns. Again, the assumption is reasonable because if players are rational they should care bout the future to maximize their overall utilities. Lastly, we assume that the players are aware of the fact that their actions are evaluated by opponents. This does not imply that the players know how they are evaluated or the result of the evaluation about their actions. The assumption holds in a real world since people know they are responsible for their actions when interacting with others unless they are absolutely sure that the others will forget about what they did.

Under these reasonable assumptions, the proposed game promotes honest cooperations even when players can choose dishonest cooperations which cost less.

Since our algorithm is distributed and does not assume any external conditions, we argue that, if the assumptions are met, our algorithm models the real world scenarios. In other words, in any distributed multi-agent systems, the true cooperation among agents eventually emerges without help of outside constraints. For example, consider a case where a well functioning distributed system exists and some intruders come and try to take advantage of the cooperative nature of the agents in the system. The intruders may get chances to take advantages at first, but soon enough the intruders get excluded from interactions because even if they do not care about their reputation or future returns, the reputation values held by other agents about them punish them eventually. Also our algorithm is applicable to large size distributed systems without overheads since the algorithm is only run privately by each agent, which also implies that we do not need to assume that other agents in the system are using the same algorithm.

CHAPTER 7
CONCLUSIONS AND FUTURE RESEARCH

In this thesis, we have introduced two algorithms to automatically compute authors' reputation from documents, a model to recursively compute reputations of nodes on social networks, called *ReMSA*, a degree estimation algorithm, and a bayesian game that models behaviors of users when there exist dishonest interactions. We have shown that our reputation computation algorithms achieve accurate results by comparing them with reputation values computed by existing methods. To relax the assumption and generalize the introduced model, *ReMSA*, we developed a degree estimation algorithm, since the degree information was assumed to be given in *ReMSA*. To be able to observe behaviors of users based on reputation values, we designed a *bayesian reputation game* where there are two types of users exist, namely, *honest* and *dishonest*. We show that *honest* users survive when the game is repeated and users value future returns.

7.1 Summary of Contributions

So far, we have discussed how to compute reputation values of users in social networks. We summarize some of the important contributions we have made.

- In Chapter 3, we introduce two existing macro-level reputation computation algorithms and apply them to compute reputation values of domain specific email authors. Then, we develop a customized time decaying function for computing reputations of email authors. Since the domain of the emails is computer security, we consider a topic is of importance for a week and is completely decayed after a week. If the same name of topic is discussed again after a week, it is considered as a new topic.

- In Chapter 4, we introduce a micro-level reputation computation algorithm called *ReMSA*. One of the distinguishing feature of *ReMSA* is that positional information of the nodes in a network is used to compute reputation values. The degree of a computing node is used to weight between its **evaluation** from direct experience and **opinions** from other users. Also, the degrees of neighbors are used to weight each neighbor's **opinion**.

- In *ReMSA*, we use a recursive voting mechanism to aggregate neighbors' **opinions** when updating reputation values. The voting process is recursive and eventually reaches to every node in the network.

- In *ReMSA*, we also consider frequencies of interactions between users to compute reputations. Some interactions occur with high frequencies and we value those interactions more than interactions with constant or decreasing speed. Using accelerations of interactions, we manipulate the **opinion** values to emphasize the significance of each **opinion**.

- In Chapter 5, we develop a distributed degree estimation algorithm that can accurately estimate neighbors' degrees. The algorithm is based on Beta distribution which

is prior and posterior of Binomial distribution. The number of observed interactions is used as the number of success in a binomial distribution.

- In Chapter 6, we study behaviors of users when dishonest users exist. We formulate a bayesian game where payoffs are private reputation values and show that cooperations among users are sustained, i.e., users act honestly, if all users are rational and value future returns.

7.2 Future Research

For the game theoretic approach, we used same future discount value for all players which implies that all the players are equally concerned about future returns. To accommodate distributed level of future expectations, we will use different discount factors to each individual to observe any behavioral change.

Also, we plan to apply our degree estimation algorithm recursively similar to the distributed voting mechanism in *ReMSA* which will make the results more comparable with global information.

We mainly focused on micro-level approaches in this thesis where there is no coordinator who manages a system and each user runs algorithms to compute information it needs, such as reputation. We want to extend our distributed algorithms where some realistic centralized environments exist. It is very hard to estimate true information without any global knowledge, if possible at all. Although, we have shown that our algorithms, such as *ReMSA* and degree estimation, can achieve relatively good performances in distributed manners, it would be interesting to apply our work to networks where some architectural restrictions exist, i.e, high level topology of a network is globally known.

REFERENCES

[1] Jason J. Jung, "Trustworthy knowledge diffusion model based on risk discovery on peer-to-peer networks.", *Expert Systems with Applications*, vol. 36, pp. 7123–7128, 2009.

[2] Jason J. Jung, "Integrating social networks for context fusion in mobile service platforms", *The Journal of Universal Computer Science*, vol. 16, no. 15, pp. 2099–2110, jul 2010.

[3] Michael Friendly, "Milestones in the history of thematic cartography, statistical graphics, and data visualization", in *13th International Conference on Database and Expert Systems Applications (DEXA 2002), AIX EN PROVENCE*. 1995, pp. 59–66, Press.

[4] J. Carbo, J. M. Molina, and J. Davila, "Trust management through fuzzy reputation", *International Journal of Cooperative Information Systems*, vol. 12, no. 01, pp. 135–155, 2003.

[5] Giorgos Zacharia and Pattie Maes, "Trust management through reputation mechanisms", *Applied Artificial Intelligence*, vol. 14, no. 9, pp. 881–907, 2000.

[6] Saeed Javanmardi, Mohammad Shojafar, Shahdad Shariatmadari, and Sima S. Ahrabi, "FRTRUST: a fuzzy reputation based model for trust management in semantic P2P grids.", *Computing Research Repository*, vol. abs/1404.2632, 2014.

[7] Yuhong Liu and Yan (Lindsay) Sun, "Anomaly detection in feedback-based reputation systems through temporal and correlation analysis", in *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, Washington, DC, USA, 2010, SOCIALCOM '10, pp. 65–72, IEEE Computer Society.

[8] M E J Newman and Juyong Park, "Why social networks are different from other types of networks.", *Physical Review E - Statistical, Nonlinear and Soft Matter Physics*, vol. 68, no. 3 Pt 2, pp. 036122, 2003.

[9] J. Golbeck and J. Hendler, "Reputation Network Analysis for Email Filtering", in *Proceedings of Conference on Email and Anti-Spam (CEAS)*, Mountain View, USA, July 2004.

[10] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler, "Yale: Rapid prototyping for complex data mining tasks", in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, Lyle Ungar, Mark Craven, Dimitrios Gunopulos, and Tina Eliassi-Rad, Eds., New York, NY, USA, August 2006, pp. 935–940, ACM.

[11] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy, "Gephi: An open source software for exploring and manipulating networks", in *International AAAI Conference on Weblogs and Social Media*, 2009.

[12] H. Ebel, L. I. Mielsch, and S. Bornholdt, "Scale-free topology of e-mail networks", *Physical Review E*, vol. 66, pp. 035103+, 2002.

[13] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data", *SIAM Review*, vol. 51, no. 4, pp. 661–703, Nov. 2009.

[14] Mark E. Newman, "Assortative mixing in networks", *Physical Review Letters*, vol. 89, no. 20, pp. 208701, 2002.

[15] Vladimir Batagelj, "Efficient algorithms for citation network analysis", *Computing Research Repository*, vol. cs.DL/0309023, 2003.

[16] Jordi Sabater and Carles Sierra, "Regret: Reputation in gregarious societies", in *Proceedings of the Fifth International Conference on Autonomous Agents*, New York, NY, USA, 2001, AGENTS '01, pp. 194–195, ACM.

[17] JooYoung Lee and Jae C. Oh, "A model for recursive propagations of reputations in social networks.", in *ASONAM*, Jon G. Rokne and Christos Faloutsos, Eds. 2013, pp. 666–670, ACM.

[18] James S. Walker, *Physics*, Addison Wesley, 4 edition, 1 2009.

[19] JooYoung Lee, Yue Duan, Jae C. Oh, Wenliang Du, Howard Blair, Lusha Wang, and Xing Jin, "Social network based reputation computation and document classification", *The Journal of Universal Computer Science*, vol. 18, no. 4, pp. 532–553, feb 2012.

[20] J. Chang, K. K. Venkatasubramanian, A. G. West, S. Kannan, I. Lee, B. T. Loo, and O. Sokolsky, "As-cred: Reputation and alert service for interdomain routing", *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1, 2012.

[21] "Routeviews", `http://www.routeviews.org`, 2010.

[22] "A random graph generator", `https://code.google.com/p/graph-generator/`, 2009.

[23] Jure Leskovec and Christos Faloutsos, "Sampling from large graphs", in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2006, KDD '06, pp. 631–636, ACM.

[24] Jian Chang, KrishnaK. Venkatasubramanian, AndrewG. West, Sampath Kannan, BoonThau Loo, Oleg Sokolsky, and Insup Lee, "As-trust: A trust quantification

scheme for autonomous systems in bgp", *Trust and Trustworthy Computing*, vol. 6740, pp. 262–276, 2011.

[25] "The caida as relationships dataset", `http://www.caida.org/data/active/as-relationships/`, 2010.

[26] Michael Hay, Chao Li, Gerome Miklau, and David Jensen, "Accurate estimation of the degree distribution of private networks", in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, Washington, DC, USA, 2009, ICDM '09, pp. 169–178, IEEE Computer Society.

[27] Tom A. B. Snijders, "Accounting for degree distributions in empirical analysis of network dynamics", in *Proceedings of the National Academy of Sciences USA*. 2003, pp. 109–114, Press.

[28] Andrey Kupavskii, Liudmila Ostroumova, Dmitry A. Shabanov, and Prasad Tetali, "The distribution of second degrees in the buckley-osthus random graph model.", *Internet Mathematics*, vol. 9, no. 4, pp. 297–335, 2013.

[29] Tianyi Wang, Yang Chen, Zengbin Zhang, Tianyin Xu, Long Jin, Pan Hui, Beixing Deng, and Xing Li, "Understanding graph sampling algorithms for social network analysis", in *ICDCS Workshops*. 2011, pp. 123–128, IEEE Computer Society.

[30] Bruno Ribeiro and Don Towsley, "Estimating and sampling graphs with multidimensional random walks", in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, New York, NY, USA, 2010, IMC '10, pp. 390–403, ACM.

[31] Christoph Lenzen and Roger Wattenhofer, "Distributed algorithms for sensor networks", *Philosophical Transactions of the Royal Society A, 370(1958)*, January 2012.

[32] Donna L. Hoffmann and Marek Fodor, "Can you measure the roi of your social media marketing?", *MIT Sloan Management Review*, vol. 52, no. 10, pp. 40–49, 2010.

[33] Réka Albert and Albert-László Barabási, "Statistical mechanics of complex networks", *Rev. Mod. Phys.*, vol. 74, pp. 47–97, Jan 2002.

[34] Shaozhi Ye and Shyhtsun Felix Wu, "Estimating the size of online social networks.", in *SocialCom/PASSAT*, Ahmed K. Elmagarmid and Divyakant Agrawal, Eds. 2010, pp. 169–176, IEEE Computer Society.

[35] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, "The pagerank citation ranking: Bringing order to the web.", Technical Report 1999-66, Stanford InfoLab, November 1999.

[36] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee, "Measurement and analysis of online social networks", in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, New York, NY, USA, 2007, IMC '07, pp. 29–42, ACM.

[37] P. Erdös and A. Rényi, "On random graphs, I", *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[38] Andrea Galeotti, Sanjeev Goyal, Matthew O. Jackson, Fernando Vega-Redondo, and Leeat Yariv, "Network games", *Review of Economic Studies*, vol. 77, no. 1, pp. 218–244, 01 2010.

[39] George Casella and Roger Berger, *Statistical Inference*, Duxbury Resource Center, June 2001.

[40] G. Csanyi and B. Szendroi, "Structure of a large social network", *Physical Review E*, vol. 69, no. 3, pp. 036131, Mar. 2004.

[41] P Shannon, A Markiel, O Ozier, N S Baliga, J T Wang, D Ramage, N Amin, B Schwikowski, and T Ideker, "Cytoscape: a software environment for integrated

models of biomolecular interaction networks", *Genome Research*, vol. 13, no. 11, pp. 2498–2504, Nov. 2003.

[42] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi, "On the evolution of user interaction in facebook", in *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[43] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee, "Measurement and analysis of online social networks", in *In Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMCÕ07)*, 2007.

[44] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks", *Physical Review E*, vol. 69, no. 2, pp. 026113, Feb. 2004.

[45] Colin S Gillespie, *Fitting heavy tailed distributions: the poweRlaw package*, 2014, R package version 0.20.5.

[46] L. Mui, M. Mohtashemi, and A. Halberstadt, "A computational model of trust and reputation", in *35th Hawaii International Conference on System Science (HICSS)*, 2002.

[47] Ruchir Gupta and Y. N. Singh, "Trust estimation and aggregation in peer-to-peer network using differential gossip algorithm", *Computing Research Repository*, vol. abs/1210.4301, 2012.

[48] Sana Hamdi, Amel Bouzeghoub, Alda Lopes Gançarski, and Sadok Ben Yahia, "Trust inference computation for online social networks.", in *TrustCom/ISPA/IUCC*. 2013, pp. 210–217, IEEE.

[49] Zheng Yan, Raimo Kantola, Gaowa Shi, and Peng Zhang, "Unwanted content control via trust management in pervasive social networking.", in *TrustCom/ISPA/IUCC*. 2013, pp. 202–209, IEEE.

[50] Petteri Nurmi, "A bayesian framework for online reputation systems.", in *AICT/ICIW*. 2006, p. 121, IEEE Computer Society.

[51] Francoise Forges and Antoine Salomon, "Bayesian Repeated Games and Reputations", CESifo Working Paper Series 4700, CESifo Group Munich, 2014.

[52] Jonathan Levin, "The dynamics of collective reputation", Discussion Papers 08-024, Stanford Institute for Economic Policy Research, 2009.

[53] F. Wang, Y. Moreno, and Y. Sun, "Structure of peer-to-peer social networks", *Physical Review E*, vol. 73, no. 3, pp. 036123, Mar. 2006.

Jooyoung Lee

CONTACT

Department of Electrical Engineering and Computer Science

Syracuse University
4-288 Center of Science and Technology
Syracuse, NY 13210

web.ecs.syr.edu\~jlee150
(315) 882-5469
jlee150@syr.edu

RESEARCH INTERESTS

Reputation Management, Bayesian Game Theory, Social Network Analysis, Multi-agent Systems, Machine Learning, Data Mining

EDUCATION

Syracuse University, Syracuse, New York, USA

Ph.D. Candidate, Computer Science (expected graduation date: August, 2014)

- Dissertation Topic:
  "Trust and Reputation: Distributed Reputation Management for Multi-agents and Game Theoretic Analysis"
- Advisor: Dr. Jae C. Oh

Korea Advanced Institute of Science and Technology, Deajeon, Korea

B.A., Computer Science, Feb, 2007

- Minor in Business Administration
- Full scholarship for 4 years

CONFERENCE PUBLICATIONS

Lee, J., Oh, J.C. 2014. Agent-Centric Second Degree Estimation within Social Networks. 2014 International Conference on Principles and Practice of Multi-Agent Systems. (Under Review)

Lee, J., Oh, J.C. 2014. **Convergence of True Cooperations in Bayesian Reputation Game**. 2014 International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM), September 24-26, Beijing, China.

Lee, J., Oh, J.C. 2013. **A Model for Recursive Propagations of Reputations in Social Networks**. 2013 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), August 25.-28, Niagara Falls, Canada.

Lee, J., Duan, Y., Oh, J.C., Du, W., Blair, H., Wang, L., and Jin, X. 2011. **Automatic Reputation Computation through Document Analysis: A Social Network Approach**. 2011 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), July 25-27, Kaohsiung, Taiwan.

JOURNAL PUBLICATIONS

Lee, J., Duan, Y., Oh, J.C., Du, W., Blair, H., Wang, L., and Jin, X. 2012. **Social Network Based Reputation Computation and Document Classification**. 2012 Special Issue of Journal of Universal Computer Science.

BOOK CHAPTER

Lee, J., Oh, J. 2014. A Distributed Reputation Computation Algorithm for Social Networks. Book Chapter in Lecture Notes on Social Networks. (Accepted)

ACADEMIC EXPERIENCE

Teaching Assistant

- Structured Programming and Formal Method **Spring '11 '12 '13 '14**
  CIS623: Graduate core course for understanding computer science logic. Introduced some softwares including *alloy, jspin* and *erigone* to check satisfiability of models. The class size is

around 60-80 students. Responsibilities include office hours and demonstrating new softwares to the class.

- Introduction to Artificial Intelligence **Fall '09 '10 '12 '13**
  CIS467/667: Undergraduate and graduate course that introduces important AI algorithms and some machine learning techniques. Class size is around 20 students. Held lab sessions for lab assignments using *Lisp*.
- Introduction to Computer Science **Spring '08 '10**
  CIS252: Undergraduate course that emphasizes recursion, data structures, and data abstraction. Around 40 students and covered lab sessions advising assignments in *Haskell*.
- Design of Operating Systems **Spring '09**
  CIS586/486: Undergraduate and graduate core course covering various topics including resource management and protection of CPU, memory, and storage, file systems, input/output, concurrent process implementation, process synchronization and networking.
- Introduction to Discrete Mathematics **Fall '08**
  CIS275: Undergraduate course on basic set theory and symbolic logic. Topics include methods of proof, mathematical induction, relations, partitions, partial orders, functions, and graphs. About 70 students, held recitations to supplement lectures.

Projects

Research Assistant

JPMorgan Chase August, 2010 - December, 2011

Project Title: Data Fusion and Visualization Project

- Data Fusion: research on how to extract information from different data sets, such as *email, RSS feed, logs*, and fuse them together to improve information quality. Classification of documents, reputation computation of authors, natural language processing.
- Visualization: information obtained from the fusion stage need to be visualized in a way that human can quickly understand without going through the data. Implemented using Java applet and used Apache server, Javascript, PHP.

Undergrad research

Cross Language Information Retrieval (CLIR)
advisor: Prof. Sung-Hyon Myaeng

- In the CLIR project, I was in charge of tagging Spanish words and categorize them according to their meanings.

Professional Experience

Sun Microsystems, Korea

Internship: Sun star (Junior System Engineer) August, 2005 - February, 2006
In charge of managing linux based server machines.

Languages

Java, Haskell, Lisp, C, Javascript, PHP, Pearl, Python, Unix shell scripts

Software

Eclipse, Gephi, RapidMiner, Weka, Visual Studio, Matlab