Generating Networks for Performance Evaluation of P2P Trust Path Search Algorithms

Thesis by

Huqiu Zhang

In partial fulfilment of the Requirements
for the Degree of
Doctor of Philosophy

Newcastle University
Newcastle upon Tyne, UK

2013
(Submitted April 29, 2013)

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Aad van Moorsel for his support through many years of a part-time PhD, offering invaluable advice and guidance, keeping me going when times were tough. It has been an honor to be his Ph.D. student.

I also would like to thank Emerson Ribeiro de Mello for the earlier Peersim implementation and for discussions on some earlier work.

I would like to acknowledge the valuable contributions the anonymous reviewers of our papers.

I also would like to thank my cousin Shi Qiao and my friends Yishi Zhao, Xiao Chen, Yingjie He, Visalakshmi Suresh, Ge Gao for making my six years in Newcastle so enjoyable.

Last but not least, I would like to thank my family for all their love and support. For my parents whose love for many years and even more coming years. For my loving husband Joris whose encouragement and support is appreciated during the final three years of this Ph.D..

Abtract

Trust has become central in computing science research. The problem of finding trust paths and estimating the trust one can place in a partner arises in various application areas, including virtual organisations, authentication systems and reputation-based trust systems. We study the use of peer-to-peer algorithms for finding trust paths and probabilistically assessing trust values in systems where trust is organised similar to the 'web of trust'.

Many empirical results demonstrate that many real-life large networks and systems are scale-free, that is, the node degree follows a power law distribution. To be able to analyse such networks, "growth algorithms" have been devised that generate scale-free networks by growing the size of the network in a manner that intuitively resembles real networks. Interestingly, generation of scale-free networks with directed arcs has not been researched extensively, especially for the case that avoids duplicate arcs as well as arcs that connect a node with itself (self loops). Being able to easily generate scale-free networks with these properties allows more accurate and efficient evaluation and simulation of routing algorithms and applications. We consider various different graph algorithms, which modify existing network generating models for directed graphs. A mathematical framework is presented to prove under which conditions the algorithm can generate networks with the scale-free feature. Since a complete proof is not feasible, we evaluate if these algorithms generate scale-free networks using statistical tests. We find that removing multiple arcs and self loops after an entire network has been generated does not affect the scale-free character, but at the cost of the growth nature of the algorithm.

To obtain reliable results with small enough confidence intervals through simulation, one needs to run many simulations and generate many networks and it is therefore of importance to generate networks with the desired properties in reasonable time. We implement a set of algorithms and compare them with respect to CPU time and memory use, in terms of both theoretical complexity analysis and experimental results. We show through experiments that using relatively standard equipment networks with a million or more nodes can be generated in mere seconds.

Finally, we explore the suitability of using peer-to-peer algorithms for finding trust paths and inferring the trust value of a set of trust paths discovered. We employ discrete event simulation and Monte Carlo techniques to evaluate these search algorithms. We implement all the relevant methods and search protocols in the Peersim simulation environment. Our main conclusion is that many peer-to-peer algorithms perform similarly, and only differ through (and are sensitive to) parameter choices, such as the number of nodes to which a query is forwarded. We also conclude that flooding is the most practical method if one stresses the requirement for finding all the trust paths, and if networks are less densely connected.

Table of Contents

List of Figures

List of Tables

Nomenclature

PDF	**P**robability **D**istribution **F**unction
CDF	**C**ulumative **D**istribution **F**unction
B2B	**B**usiness to **B**usiness
P2P	**P**eer to **P**eer
PGP	**P**retty **G**ood **P**rivacy
LSCC	**L**argest **S**trong **C**onnected **C**omponent
TTL	**T**ime to **L**ive
EER	**E**nhanced **E**ntity **R**elationship
KRR	**K**rapivsky, P. L., **R**odgers, G. J. and **R**edner, S.
CSN	**C**lauset, A., **S**halizi, C. R. and **N**ewman, M. E. J.

To my parents

&

my husband

Chapter 1

Introduction

1.1 Background

To motivate our work, consider a possibly large number of people or businesses that want to collaborate, and not all players know each other. The internet and business-to-business (B2B) technologies promise a world in which such collaborations can be created almost instantly (called virtual organisations). One of the challenges in creating such dynamic business interactions is the establishment of trust, and assume therefore that each party maintains a list of trusted parties, including a parameter quantifying the amount of trust placed in a party. In that situation, parties may decide to trust each other and initiate business if a path of trust relations exists between them (in both directions), and they may calculate risks and decide about their actions depending on the trust values associated with these paths.

Our trust model views the system similar to the Web of Trust, a network where nodes are linked if they have a trust relation. We assume links are directed, that is, a link or arc from A to B implies that A trusts B, but B does not necessarily trust A. In our model, with each link is associated a value to represent the amount of trust associated to the trust relation. This value shows the strength of the trust relation. Every node maintains its trust relations associated with other nodes, and since not all nodes of a network have direct interactions, trust links do not exist between all pairs. If a node does not have a trust relation with some other node, it can estimate trust indirectly. For instance, assume there is no arc from A to C, A can still establish trust in C if there exists at least one path from A to C in the graph. This implies that trust is transitive in such a model. Given a pair of request node and target node, which have no established links, the request node can estimate the overall trust value, which is calculated on a set of discovered trust paths between them, if existing. We will discuss this in Chapter 6.

Due to the similarity of the structure of a web of trust network and an unstructured peer-to-peer network, peer-to-peer search algorithms can be applied to discover trust paths and probabilistically assess trust values. We employ a model-based evaluation of peer-to-peer search algorithms in the context of the web of trust. First a network model is considered to produce networks that have properties similar to web of trust networks. Many empirical results demonstrate that many real-life large networks and systems are scale-free, that is, the node degree distribution follows a power law distribution. It has been shown that the web of trust network also exhibits a scale-free nature (Guardiola et al., 2002). Interestingly, generation of scale-free networks with directed arcs (as needed in the web of trust) has not been researched extensively, especially for the case that avoids duplicate arcs as well as arcs that connect a node with itself (self loops). Therefore, we derive such algorithms in this dissertation.

1.2 Aims and Objectives

Our specific goals are as follows.

Firstly, we aim to address the generation of directed scale-free network to reproduce the scale-free nature of real-life networks. We consider graph algorithms that modify an existing “growth algorithms”. Growth algorithms increase the size of the generated network in each iteration according to same rules. Growth algorithms are attractive because intuitively they match the way real networks evolve. We aim to present a mathematical framework to prove under which conditions the algorithm can generated networks with the scale-free feature. Since a complete proof is not feasible, we also aim to evaluate if these algorithms generate networks with the scale-free feature using statistical tests.

Secondly, we aim to develop fast implementations of the algorithm. To obtain reliable results with small enough confidence intervals through simulation, one need to run many simulations and generate many networks and it is therefore of importance to generate networks with the desired properties in reasonable time. We aim to implement a set of algorithms and compare them with respect to CPU time and memory use, in terms of both theoretical complexity analysis and experimental results.

Finally we aim to analyse how peer-to-peer search algorithms perform when applied to finding trust paths and calculating trust values through simulations. We collect existing peer-to-peer search algorithms and employ discrete event simulation and Monte Carlo techniques to evaluate these search algorithms. The metrics are defined that would be used in the simulations. The simulation methodology including the sampling method and the trust computation method is considered.

1.3 Contributions

Here we highlight the contributions of our research work.

1. A set of growth algorithms for generating scale-free directed networks without multiple arcs and self loops.
   A class of growth algorithms are developed to generate scale-free networks by growing the size of the network in a manner that intuitively resembles real networks.

2. An efficient implementation of the growth algorithms.
   In order to generate sufficiently large networks for simulation, with the desired characteristics in reasonable time, we design and implement a set of algorithms. These algorithms are compared with respect to CPU and memory use, using both theoretical complexity analysis and experimental results. The derived approach allows networks with a million or more nodes to be generated within seconds on current-day desktops.

3. A non-growth but attractive algorithm which generates directed scale-free networks.

A non-growth algorithm is proposed to address the generation of directed scale-free networks to reproduce the scale free feature of real-life networks without multiple arcs and self loops.

4. A mathematical proof of the conditions for scale-freeness of algorithms.
   For our set of growth algorithms, this mathematical framework enables one to prove if the network algorithm can generate networks with the scale free feature. It shows the conditions under which growth algorithms are power law. A corrected proof of one given with is also presented to prove the power law distribution of the generated networks for an existing growth algorithm.

5. Statistical test of power law distribution of all algorithms.
   Quantitive statistical tests are conducted on all network generating algorithms with respect to power law hypothesis. The extensive tests are carried out under different circumstances, such as taking into account various parameters values. We also study how the scale-free feature evolves with the network growth.

6. Evaluation of P2P algorithms in the context of web of trust.
   We explore the suitability of using peer-to-peer search algorithms for finding trust paths and probabilistically assessing trust values in systems where trust is organised similar to the Web of Trust. We do this through discrete event simulation of random as well as scale-free trust networks based on flooding as well as selective search algorithms. The simulation methodology including the sampling method and the trust computation method is defined.

1.4 Structure of Thesis

The structure of this thesis is as follows: it starts with a general discussion and definitions of trust, web of trust, followed by a survey of network generating models, and reputation based trust systems in peer-to-peer networks from the literature. The difference with the work in this thesis is also stated.

In Chapter 3, we explore the possibility of finding an algorithm that allows us to generate directed scale-free networks that do not have multiple arcs or self loops.

It concerns some growth algorithms and a non-growth algorithm, which are the modified versions of an existing network generating algorithm. Then we present a generic mathematical framework for proving the power law of all algorithms.

In Chapter 4, we analyse all the algorithms by presenting statistical test results for these algorithms. We also present the statistical test results of the empirical data which is collected from the Wotsap Web of Trust project.

Chapter 5 then considers how to efficiently implement our growth algorithms, which allows the network generating model to be effectively used in discrete-event studies. We design and implement a set of algorithms and compare them with respect to CPU and memory use, in terms of both theoretical complexity analysis and experimental results.

Having established the underlying network, we turn to study the suitability of using peer-to-peer algorithms for discovering trust paths and inferring the trust value of a set of trust paths in the context of web of trust in Chapter 6. We employ discrete event simulation and Monte Carlo techniques to evaluate the performance of variations of flooding search algorithm, in random as well as scale-free networks. We also analyse the performance and compare the cost, considering the message overhead and the success rate.

1.5 Publication History

Here we will list the publications of the work in this thesis.

- Huqiu Zhang and Aad van Moorsel. Evaluation of P2P Algorithms for Probabilistic Trust Inference in a Web of Trust. *Computer Performance Engineering: 5th European Performance Engineering Workshop*, pages 242-256, 2008.

- Huqiu Zhang and Aad P. A. van Moorsel. Fast Generation of Scale Free Networks with Directed Arcs. *Computer Performance Engineering: 6th European Performance Engineering Workshop*, pages 131-148, 2009

The work in Chapter 3 and Chapter 4 has not been published yet.

Chapter 2

Background And Literature Review

This chapter begins with a general discussion and definitions of trust, web of trust and reputation based trust systems from the literature. Then we present an overview of research areas and references in network generating models.

2.1 Trust

The study of trust spans several disciplines, including sociology (Luhmann, 1988), psychology (Bulter and Cantrell, 1984), political science (Williamson, 1993), business and economics (Granovetter, 1985; Husted, 1989). In order to give the reader a reference point for understanding the trust, we present a list of general definitions cited from existing research.

The Oxford English Dictionary defined trust as "confidence in or reliance on some quality or attributes of a person or thing, or the truth of a statement."

Sociologist Coleman (1990) defined trust as "an incorporation of risk into the decision of whether or not to engage in the action" by acting based on estimates of the likely future behaviour of others.

Mcknight and Chervany (1996) defined trust as "the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible."

Grandison and Sloman (2000) described trust as "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context. "

In this thesis we adopt the following definition proposed in (Mahoney et al., 2005): "trust is one's reasonable expectation of a positive outcome in a situation where there is less than full control over the actions of the participants."

Typically trust is established between two parties based on direct interactions between them. This type of trust is called direct trust (Boeyen et al., 2004; Mahoney et al., 2005). Direct trust is obtained without "reliance on intermediaries"(Boeyen et al., 2004). As pointed out in (Jøsang and Pope, 2005), trust can be transitive with certain semantic constraints. A simplified description of the concept of transitive trust is that "assume that agent A trusts agent B, and that agent B trusts agent C, then by transitivity, agent A trusts agent C" (Jøsang and Pope, 2005). The trust agent A places on agent C is viewed as indirect trust (Boeyen et al., 2004; Mahoney et al., 2005), which is derived from pre-existing trust established with intermediaries. Also note that, as pointed out in (Mahoney et al., 2005), trust relations usually are one-way: agent A trusts agent B does not mean agent B trusts agent A.

As trust is complex, personal, subjective and it changes over time, trust may be quantified and computed in many different ways. As being represented that a claim is true or false, a binary value is used in industry specifications, for instance WS-Trust (Anderson et al., 2005) and the OASIS trust model guidelines (Boeyen et al., 2004). Some approaches employ discrete values to measure the level of trust. For example, in the PGP system (Zimmermann, 1995), the degree of trust can be specified unknown, partial trust or complete trust. In (Carbone et al., 2003) degrees of trust on an entity are described by a set of discrete values: low, medium, or high. Some other approaches choose a continuous numerical range. As an example, in (Jøsang, 2001) a subjective belief represented by a probability results in a real number between 0 and 1.

2.2 Web of Trust

The Web of Trust was introduced by Phil Zimmermann in 1992, as a concept used in Pretty Good Privacy (PGP) (Zimmermann, 1995), the GNU Privacy Guard (GnuPG) and openPGP systems (Callas et al., 2007). In such systems any identity can verify another identity without the need of a Certification Authority (CA). In public key systems, the CA is a trusted third party, which issues digital certificates to authenticate the ownership of the public key. In the web of trust, instead of using CA, each user is associated with a public/private key pair and can issue certificates to others by signing others' public key certificates with their own private key. There are two trust categories in the PGP trust model: the trustworthiness of the public key certificate and the trustworthiness of an introducer (Abdul-Rahman, 1997). The trustworthiness of the public key certificate is about whether a public key certificate is valid or not, that is, whether the public key actually belongs to the person who claims it. The trustworthiness of an introducer is about how much one can trust the introducer to be a competent signer of a public key certificate.

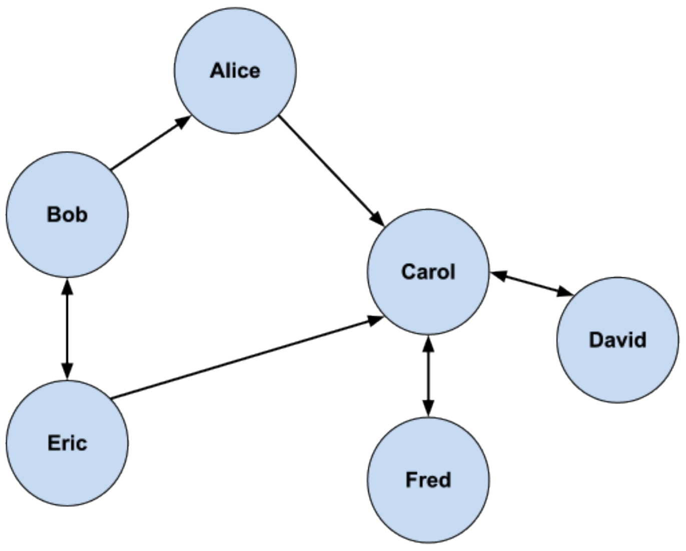

Figure 2.1: An example of the PGP web of trust

As an example of the PGP trust model, shown in Figure 2.1, nodes are public keys which represent users and arcs are signatures. In particular, a directed arc from Alice to Carol represents Alice signing a certificate which binds Carol to a particular public key. When doing so, Alice checks identity papers of Carol (e.g.

a passport), one can say that Alice trusts Carol, or more specifically, Alice trusts that Carol is who she claims she is. We set up a scenario where Bob wants to communicate with Carol whom he never met before. Assume that Bob knows Alice and signed Alice's public key. If Bob receives Carol's public key certificate, Bob does not know Carol, however Bob checks the signatures on the certificate and sees if he trusts any of them. Based on this information Bob can decide if he wishes to trust Carol's public key certificate. Bob sees Alice is among Carol's public key certificate signers, one can imagine therefore, Bob can be confident that Carol's public key is authentic. At the current stage, a trust path is formed from Bob to (the key of) Carol, which implies that Bob extends trust to the new party. Furthermore, there are two trust paths from Bob to David: one list of keys $Bob \rightarrow Alice \rightarrow Carol \rightarrow David$ where Bob signed Alice, Alice signed Carol, etc. That means Bob trusts Alice's public key certificate, Alice trusts Carol's public key certificate, and so on. Another list of keys is $Bob \rightarrow Eric \rightarrow Carol \rightarrow David$ where Bob signed Eric, Eric signed Carol, Carol signed David. In this trust path, Bob trusts Eric's public key certificate, Eric trusts Carol's public key certificate, and so on. Based on these intermediaries, Bob may trust that the public key labeled 'David' actually belongs to 'David'. All these trust paths forms what is called the web of trust. In this thesis our system is similar to the web of trust.

In the work of analysing the web of trust's structure, the Wotsap project (Cederlof, 2007) keeps daily track of the largest strongly connected set of OpenPGP keys. More recently, Ulrich et al. (2011) present a thorough analysis of the OpenPGP web of trust with a focus on determining various properties, for instance, scale-free property, random removal of nodes, and community structure.

2.3 Peer-to-Peer Systems

To address the problem of finding one or more trust path, in (Ribeiro de Mello et al., 2007), the authors indicate that peer-to-peer search algorithms can be applied to discover trust paths, due to the similarity of the structure of a web of trust network and an unstructured peer-to-peer network. Many definitions regarding peer-to-peer (P2P) systems have been presented in the literature. The commonly cited definition of peer-to-peer system is due to (Androutsellis-Theotokis and Spinellis, 2004): "Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority". Without the need for central coordination by servers, every node in a P2P network is both a supplier and a consumer of resources. Due to resource optimisations it enables on network efficiency, scalability and stability, the decentralised nature results in P2P systems widely used in various areas: file sharing networks (Gnutella, 2001; *Vuze*, 2003), communications networks (voice over P2P application, (*Skype*, 2003), Instant messaging), science (bioinformatics). P2P systems can be divided into several classes (Androutsellis-Theotokis and Spinellis, 2004).

Centralized P2P systems Centralized P2P systems utilize a single central server to manage the system. Napster is an example of centralized P2P system, where a central server contains "a peer list and content directory" and is responsible for "peer discovery and content lookup" (Lui and Kwok, 2002). The problem with such systems is that it has a single point of failure. If an attack has been used to shut down the central server, the system becomes useless.

Decentralized structured P2P systems Decentralized structured P2P systems have no central server, however these systems use structured overlays to organize nodes and data items. A structured overlay means "the overlay topology is tightly controlled and files are place at precisely specified locations" (Androutsellis-Theotokis and Spinellis, 2004). In another way to say, content objects are not placed to random nodes. In such systems, every node is assigned a unique identifier and every data item is assigned a unique identifier. A distributed hash table (DHT) is used to map a data item to a particular peer based on its ID, both whose IDs fall into the same ID space. Structured systems provide more efficient data lookup, compared to unstructured P2P systems (Shen et al., 2009). Chord (Stoica et al., 2001) is a well-known DHT-based P2P system.

Partly-decentralized P2P systems Partly-decentralized P2P systems employ an hierarchical overlay, which consists of ordinary nodes and super nodes. The selection of super nodes is concerned with high capacities (bandwidth, storage space and computing power). Each supernode maintains indices of local nodes as well as the information of other super nodes. The whole system can be viewed as a set of clusters of “ordinary peers with a single super peer”. Content object searches are conducted on the super node level. An ordinary peer sends a search query to its super peer. If content object is not found, the super peer forwards the search query to other connected super peers. Kazaa (*Kazaa*, 2006) is a typical example of this kind of systems.

Decentralized unstructured P2P systems Different from structured systems, an unstructured P2P system has the overlay topology which ”the placement of content is completely unrelated to”. In such systems, data are distributedly stored in the nodes and each node independently connects to a few other nodes. Gnutella (Gnutella, 2001) is an example of unstructured P2P systems for file sharing. File search in Gnutella is based on flooding. A request node sends a search query to all its neighbours. If a node receives a search query, it forwards the query to all its neighbours.

2.4 Reputation Based Trust Systems

Given the distributed application architecture, the performance of P2P systems depends on the collaboration among distributed peers. Most P2P systems are open system, where peers often interact with unknown peers. Therefore, P2P systems need to consider the misbehavior and attacks from selfish and malicious nodes. A node in an open P2P system needs to trust other unknown nodes to exchange information and share files. Due to the autonomous and distributed nature, a number of trust issues occur in P2P networks (Wallach, 2003). The importance of mutual trust in P2P systems has been increasingly recognised. To solve this problem, trust management in P2P systems are required in order to enable peers to build trust on other unknown peers to cooperate, as well as avoid untrustworthy peers and reduce risks.

Several reputation-based trust systems for P2P network have been proposed in the literature, such as CORE (Michiardi and Molva, 2002), EigenTrust (Kamvar et al., 2003), PeerTrust (Xiong and Liu, 2004), TrustGuard (Srivatsa et al., 2005), Scrivener (Nandi et al., 2005), P2PRep (Aringhieri et al., 2006), Credence (Walsh and Sirer, 2006), and PowerTrust (Zhou and Hwang, 2007). EigenTrust, PeerTrust, and PowerTrust are all considered for Distributed Hash Table based P2P system rather than unstructured P2P system. In reputation-based trust systems, a peer is assigned a reputation value, based on the transactions it performed and feedbacks from other peers. The reputation value is used to build trust among peers.

In (Hoffman et al., 2007), the authors present an analysis framework for reputation systems, classify the known and potential attacks and describe defense strategies. Our work differs from existing research, in that we do not consider attacks, but discuss the performance of P2P algorithms for trust inference. Moreover, compared with existing research on performance evaluation of P2P algorithms (e.g. (Lv et al., 2002)), our work differs by considering trust values and multiple paths to the target.

2.5 Network Generating Models

The random graph model is introduced by mathematicians Pal Erdős and Alfred Rényi (Erdős and Rényi, 1959), who assumed that complex networks were wired randomly together. This hypothesis motivated extensive work on the study of complex networks to discover the general features of real complex networks. This includes the world wide web (Broder et al., 2000), citation network (Lehmann et al., 2003), e-mail network (Ebel et al., 2002), web of trust (Guardiola et al., 2002), Wikipedia (Holloway et al., 2005), and the empirical results demonstrate that many real-life large networks have the scale-free topological property, which means that the number of arcs in a node is power law distributed. Such networks with this typical feature are called scale-free networks. We discuss this point thoroughly in Section 3.2.

Barabási and Albert (Barabasi and Albert, 1999) first proposed an algorithm for generating networks with a power-law degree distribution. In the Barabási-Albert model a network grows by adding new nodes to the system with a constant rate, and any new node is preferentially attached to the m old ones that already have a large number of links at the moment. The scale-free nature of the resulting network indicates that growth and preferential attachment reproduce the scale-free feature observed in real networks. Growth here means that the number of nodes in a network increases over time. Preferential attachment means that the nodes that already have a large number of links are more likely to acquire new links (the “rich-get-richer” effect) (Caldarelli, 2007). The limit of the Barabasi-Albert model is that due to its construction, each node has exactly m outgoing edges, only the incoming degree distribution follows a power law.

Among the network generating models, growth network models are attractive and popular, as the growth and preferential attachment is believed to exist widely in real networks. Many modifications and alternatives to the Barabási-Albert model have been developed. In (Caldarelli, 2007), the author classifies models into three categories: fitness based model, ageing process model and copying model. The fitness based model introduces a fitness effect (Bianconi and Barabási, 2001), where each node has a specific feature called fitness, which the preferential attachment takes into account. This fitness idea has been considered in various ways, e.g. (Goh et al., 2001), (Caldarelli et al., 2002), and (Boguñá and Pastor-Satorras, 2003).

The ageing process model introduces an ageing effect (Klemm and Eguíluz, 2002). At a particular time a node can take one of two states, active or inactive. The active nodes can still receive edges and modify their states, while the inactive nodes are removed from the system over time.

In copying models, new nodes are attached to the nodes that are ‘copied’ as neighbours from a randomly chosen node. This mechanism can be found in two different context examples: Aiello et al. (2000) studied a graph generating model to capture some behaviour of data in telecommunications. Vázquez et al. (2003) proposed a graph generating model to represent protein interaction networks. Other similar models were proposed by Berg et al. (2004) and Goh et al. (2005).

Most above mentioned models generate scale-free networks with undirected links. For networks with directed arcs, we are aware of only two models, which are almost identical (Bollobás et al., 2003) and (Krapivsky et al., 2001). Both models can be viewed as modifications to the Barabási-Albert model, which allow new arcs to be added between existing nodes. Both models allow multiple arcs (duplicate arcs with the same origin and destination) and self loops (arcs that connect a node with itself) in the process of growing a network. Our study differs from their work in that it aims to generate directed scale-free networks without multiple arcs and self loops.

Similar to (Krapivsky et al., 2001), Capocci et al. (2006) considered the different growth mechanism to be able to reproduce the features of the on-line encyclopedia Wikipedia by means of a statistical model. Recently, Fraiman (2008) studied a particular growing directed network model where at each time step a new node with K_{out} out-going edges joins the network, where K_{out} is a Poisson-distributed random variable.

Chapter 3

Algorithms for Generating Directed Scale-Free Networks

3.1 Introduction

There exist a large number of algorithms to generate scale-free networks with undirected links (see Section 2.5). For networks with directed arcs, however, we are aware of only two algorithms, which are almost identical: Bollobás et al. (2003) and Krapivsky et al. (2001). Both Bollobás et al. (2003) and Krapivsky et al. (2001) show that the network algorithms they develop generate networks with the distributions of in-degree and out-degree following a power law. Networks generated by using the above algorithms contain duplicate arcs with the same origin and destination ('multiple arcs') and with the destination as the same as the origin ('self loops'). Note that we will base our work on the specific algorithm from (Krapivsky et al., 2001) and named it as the KRR algorithm by the algorithm creators' names.

This chapter sets out a study of finding an algorithm that generates directed scale-free networks that do not have multiple arcs or self loops. The absence of self loops and multiple arcs is crucial for many real life applications as, for instance, it does not make sense to imagine a node in a trust network to have a trust relationship with itself. Being able to easily generate scale-free networks with these properties allows more accurate and efficient evaluation and simulation of routing algorithms and application protocols, such as in the trust path discovery in Chapter 6.

The key contributions of the chapter are: a set of growth algorithms for generating directed networks without multiple arcs and self loops, a non-growth but attractive algorithm which generates directed scale-free networks, a corrected mathematical proof of the KRR algorithm, and a mathematical framework for determining the conditions under which the degree distribution is power law.

The rest of this chapter is organized in the following manner: we first present an existing network generating algorithm that generates directed scale-free networks (contain multiple arcs and self loops). In Section 3.3 we propose growth and non-growth algorithms, which are the modified versions of the existing network generating algorithm. All the algorithms are concerned to avoid multiple arcs and self loops in a resulting network. In Section 3.4 we introduce a mathematical generic framework to prove under which conditions the algorithm can generate networks with the scale-free feature.

3.2 Scale-Free Networks

The key property of the algorithms we generate here is scale-freeness. We will therefore give an intuitive and a mathematical description of this property. Intuitively, scale-free networks are networks where the vast majority of nodes possess very few links but a few nodes possess many links. When, for instance, thinking of a university website, the main homepage would be highly connected since there are links from many individual pages towards it directly and it links to many pages itself. Conversely many individual pages may only contain a link or two.

Mathematically a standard definition (Caldarelli, 2007) of scale-free networks is that the distribution over the node degrees follows a power law form. A nonnegative random variable X is said to have a power law distribution if the

$$\Pr(X = x) \propto x^{-\alpha}, \tag{3.1}$$

where $\propto$ means "is proportional to", α is a constant parameter. In directed scale-free networks, both in-degree distribution and out-degree distribution may follow the power law distribution.

Based on empirical observations from many real-life networks, power law applies in the degree values that are bigger than some minimum degree x_{min}, and α lies between 2 and 3. The reason behind can be explained by looking at the mean value and the variance value of number of arcs that a node has in the network. In this case, the mean value is given by taking the product of each possible value of x and its probability $P(x)$, and then adding all these products together, giving $E(X) = \sum xP(x)$. In probability theory and statistics, the computational formula for the variance $Var(X)$ of a random variable X is the formula $Var(X) = E(X^2) - [E(X)]^2$, where $E(X)$ is the expected value of X. Considering the above probability formula, we have

$$E(X) = \sum_{x=1}^{\infty} x\, P(x) = \sum_{x=1}^{\infty} x\, x^{-\alpha} = \sum_{x=1}^{\infty} x^{1-\alpha}, \tag{3.2}$$

and

$$E(X^2) = \sum_{x=1}^{\infty} x^2\, P(x) = \sum_{x=1}^{\infty} x^2\, x^{-\alpha} = \sum_{x=1}^{\infty} x^{2-\alpha}. \tag{3.3}$$

$$\begin{aligned}
\alpha = 2, \quad & E(X) = \sum_{x=1}^{\infty} x^{1-2} = \sum_{x=1}^{\infty} \frac{1}{x} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots = \infty \\
\alpha = 2, \quad & Var(X) = E(X^2) - [E(X)]^2 = \sum_{x=1}^{\infty} x^{2-2} - \left(\sum_{x=1}^{\infty} x^{1-2}\right)^2 \\
& = \sum_{x=1}^{\infty} 1 - \left(\sum_{x=1}^{\infty} \frac{1}{x}\right)^2 = \sum_{x=1}^{\infty} 1 - \sum_{x=1}^{\infty}\sum_{y=1}^{x} \frac{1}{x}\frac{1}{y} \\
& = \sum_{x=1}^{\infty} \left[1 - \frac{1}{x}\left(\sum_{y=1}^{x} \frac{1}{y}\right)\right] \geq \sum_{x=1}^{\infty} \left(1 - \frac{1}{2}\right) = \sum_{x=1}^{\infty} \frac{1}{2} \\
\alpha = 3, \quad & E(X) = \sum_{x=1}^{\infty} x^{1-3} = \sum_{x=1}^{\infty} \frac{1}{x^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots = \frac{\pi^2}{6} \\
\alpha = 3, \quad & Var(X) = E(X^2) - [E(X)]^2 = \sum_{x=1}^{\infty} x^{2-3} - \left(\frac{\pi^2}{6}\right)^2 = \infty \\
\alpha = 4, \quad & E(X) = \sum_{x=1}^{\infty} x^{1-4} = \sum_{x=1}^{\infty} \frac{1}{x^3} = 1 + \frac{1}{2^3} + \frac{1}{3^3} + \ldots \approx 1.202
\end{aligned}$$

$$\alpha = 4, \quad Var(X) = E(X^2) - [E(X)]^2 = \sum_{x=1}^{\infty} x^{2-4} - (1.202)^2 \approx 0.2$$

From the above calculations we can see that the value of α can only be in a range of [2, 3] to guarantee the expected number of arcs of a node is finite and the variance is infinite.

Note that a mathematical proof is about proving a power law distribution in the infinite network. This is not sufficient if you have finite networks. We do a statistical test in Chapter 4 to determine if a finite network is likely to be sampled from a power law distribution.

3.3 The KRR Algorithm

In this chapter we consider four different algorithms modified versions of the KRR algorithm. We introduce the following notation to represent the growing network generated using these algorithms. A directed network grows by adding an arc at discrete time steps. At each time step a node may or may not be added. Let G_0 be the initial directed graph. At each time step, there are n ($n \geq 1$) nodes and a set of arcs. We write $i \rightarrow j$ to denote a directed arc from node i to node j, $i, j \leq n$ and $i \not\rightarrow j$ if no arc $i \rightarrow j$ exists. The algorithm terminates when there are N nodes, with $N > 0$ some predefined target size of the generated network.

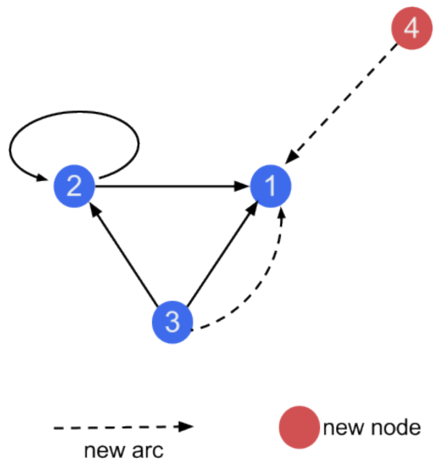

Figure 3.1: An example of the process of a growing network generated by the KRR algorithm. At a particular time step, with probability p, a new node 4 is attached to node 1, otherwise, with probability $q = 1 - p$, a new arc is added from node 3 to node 1.

We turn now to the KRR algorithm, which generates directed scale-free networks with multiple arcs and self loops. The main idea of the KRR algorithm is as follows. Starting from a small network (or a single node), the algorithm adds one arc at discrete time steps. At each time step with probability p, the algorithm adds a node and one arc, which runs from the new node to an existing node, while with probability $q = (1 - p)$, an additional arc is added between two existing nodes. Figure 3.1 illustrates the process of a growing network generated by the KRR algorithm. If a new arc is added, it will be from node i to node j depending on the out-degree value of the origin node i and in-degree value of the destination node j. In particular, for any node i, let I_i be the in-degree of node i and O_i the out-degree of node i, then the probability of adding an arc from node i to node j is proportional to $(\mu + O_i)(\lambda + I_j)$, where λ and μ are constants: $\lambda > 0$ and $\mu > -1$. This algorithm's rules are presented in the pseudocode shown in Algorithm 1.

Algorithm 1 KRR algorithm main()

```
Require: a network $G_0(V_0, E_0)$, $n = |V_0|$, $n \geq 1$
Ensure: $n = N$
 1: repeat
 2:     $a \Leftarrow random[0, 1)$
 3:     if  $a < p$ then
 4:         add a new node $m$
 5:         choose node $j$ with a probability that is proportional to weights $w_i$
 6:         generate an arc $m \to j$
 7:         update node $m$ out-degree and weight: $O_m = O_m + 1$, $v_m = \mu + O_m$
 8:         update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
 9:         increase network size by 1: $n = n + 1$
10:     else
11:         choose node $i$ with a probability that is proportional to weights $v_i$
12:         choose node $j$ with a probability that is proportional to weights $w_i$
13:         generate an arc $i \to j$
14:         update node $i$ out-degree and weight: $O_i = O_i + 1$, $v_i = \mu + O_i$
15:         update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
16:     end if
17: until there are $N$ nodes: $n = N$
```

Algorithmically, introduce, in the n-th iteration for $i = 1, ..., n$ the following weights:

$$\begin{aligned} v_i &= \mu + O_i \\ w_i &= \lambda + I_i. \end{aligned} \tag{3.4}$$

Then generate an arc $i \rightarrow j$ by conducting weighted random sampling using the weights v_i to determine i and by conducting weighted random sampling using the weights w_i to determine j. In particular, the origin node is chosen with a probability that is proportional to its weight v_i at that moment and the destination node is chosen with a probability that is proportional to its weight w_i at that moment. The algorithm terminates when the generated network reaches the predefined target network size N. The algorithm introduces self loops and multiple arcs. A self loop is an arc which connects a node to itself. Multiple arcs are more than one arcs directed between the same two nodes.

3.3.1 A corrected mathematical proof

In (Krapivsky et al., 2001), the authors presented the analysis of the degree distribution of the KRR algorithm and we base our partial proofs in this section on this definition. To give the reader a clear understanding, we first introduce their joint degree distribution equation, and then show our accurate equation for the joint degree distribution. The accuracy of our equation is verified by showing that the same theorem in (Krapivsky et al., 2001) can be proved through our equation.

To verify the algorithm to generate networks with the power law node degree distributions, Krapivsky et al. (2001) started their analysis by determining the joint degree distribution. We introduce some basic definitions and assumptions used in (Krapivsky et al., 2001). $N(t)$ is defined as the total number of nodes in the network at time t, $K(t)$ and $L(t)$ are defined as the sum of all in-degrees and the sum of all out-degrees at time t, respectively. $A(t)$ is defined as the number of arcs in the network at time t. $N_{kl}(t)$ is defined as the average number of nodes with in-degree k and out-degree l at time t. According the network growth process in the KRR algorithm, at each time step, the network grows by adding an additional arc and/or a new node. As a result, $K(t) = L(t) = A(t) = t$,

$N(t) = pt$. They derive the degree distribution $N_{kl}(t)$ according to the rate equations

$$\begin{aligned}\frac{dN_{kl}}{dt} &= (p+q)\,[\frac{(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}}{K+\lambda N}] \\ &\quad +q\,[\frac{(l-1+\mu)N_{k,l-1}-(l+\mu)N_{kl}}{L+\mu N}] + p\,\delta_{k0}\delta_{l1}. \end{aligned} \tag{3.5}$$

According to their explanation of the above equation, the first group of terms on the right-hand side are the changes in the in-degree of destination nodes. A new directed arc is added to a target node whose in-degree is k with a probability proportional to $(k+\lambda)N_{kl}$. The denominator is simply the total normalization factor $\sum_{k,l}(k+\lambda)N_{kl} = K+\lambda N$, where the total in-degrees $K = \sum_{k,l} kN_{kl}$ and out-degrees $L = \sum_{k,l} lN_{kl}$. Similarly, the second group of terms accounts for changes in out-degree. And the last term stands for changes in new nodes whose in-degree is 0 and out-degree is 1.

The mistake in Equation (3.5) is that it does not reflect self-loops. When a self loop of a node m is added, the values of both in-degree and out-degree of node m are increased by 1, and hence, a self-loop implies a node with both in-degree and out-degree changes, and this part is missed in Equation (3.5). Here we will give the accurate equation for the joint degree distribution.

Lemma 3.1. *The correct expression for* $\dfrac{dN_{kl}}{dt}$ *is*

$$\begin{aligned}\frac{dN_{kl}}{dt} &= p\,\frac{(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}}{A+\lambda N} + p\,\delta_{k0}\delta_{l1} \\ &\quad +q\,\frac{(k-1+\lambda)(l-1+\mu)N_{k-1,l-1}-(k+\lambda)(l+\mu)N_{kl}}{(A+\lambda N)(A+\mu N)} \\ &\quad +q\,\frac{[(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}][A+\mu N-(l+\mu)]}{(A+\lambda N)(A+\mu N)} \\ &\quad +q\,\frac{[(l-1+\mu)N_{k,l-1}-(l+\mu)N_{kl}][A+\lambda N-(k+\lambda)]}{(A+\lambda N)(A+\mu N)}. \end{aligned} \tag{3.6}$$

Proof of the Lemma 3.1. To make the analysis process straightforward, we analyse how in-degree and/or out-degree changes occur when a new directed arc is added in one of three situations at each time step:

- case 1: a directed arc is added from a new node to an existing node

- case 2: a directed arc is added between two existing nodes and this arc is a self loop
- case 3: a directed arc is added between two existing nodes and this arc is not a self loop

Now we need to understand how the degree distribution $N_{kl}(t)$ evolves in each situation. We introduce a probability r, defined as the probability that given an arc is added, it is a self-loop:

$$\begin{aligned} r &= Prob\{\text{the arc is a self-loop} \mid \text{an arc is added}\} \\ &= \frac{Prob\{\text{the arc is a self-loop } \wedge \text{an arc is added}\}}{Prob\{\text{an arc is added}\}} \\ &= q\,\frac{\displaystyle\sum_{n_{kl}}(k+\lambda)(l+\mu)}{\displaystyle\sum_{k,l}(k+\lambda)(l+\mu)}, \end{aligned} \tag{3.7}$$

where n_{kl} is any node with in-degree k and out-degree l. The prefactor q here is the probability of adding an additional arc between existing nodes.

Case 1: with probability p, a new node is introduced and attached to one existing node. In this situation, one existing node's in-degree changes and the new node's out-degree changes. The change details are presented in the table 3.1.

The degree type	Change
only in-degree	$p\,\frac{(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}}{\sum_{k,l}(k+\lambda)N_{kl}}$
only out-degree	$p\,\delta_{k0}\delta_{l1}$
in- and out-degrees	0

Table 3.1: in- and/or out-degree changes occur in the network in case 1

In case 1, the addition of a new node to the network leads to an increase in the number of nodes with in-degree 0 and out-degree 1. The addition of a new directed arc to a node with in-degree k leads to a loss in the number of such nodes. This occurs with a probability $(k+\lambda)N_{kl}$, divided by the denominator $\sum_{k,l}(k+\lambda)N_{kl}$. We obtain $\sum_{k,l}(k+\lambda)N_{kl} = A + N\lambda$, where A is the number of arcs and N is the number of nodes in the network at that time as mentioned above.

Case 2: with probability r, a new directed arc is added to the network, and this arc is a self-loop. In case 2, a node's in-degree as well as out-degree changes. The addition of a self-loop to a node with in-degree k and out-degree l leads to a loss of such nodes. The change details are presented in the table 3.2.

The degree type	Change
only in-degree	0
only out-degree	0
in- and out-degrees	$r\,\dfrac{(k-1+\lambda)(l-1+\mu)N_{k-1,l-1}-(k+\lambda)(l+\mu)N_{kl}}{\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}$

Table 3.2: in- and/or out-degree changes occur in the network in case 2

Case 3: a new directed arc is added to the network, and this arc is not a self-loop. Then the changes that occur in the network are presented in the table 3.3.

The degree type	Change
only in-degree	$(q-r)\dfrac{[(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}]\,[A+\mu N-(l+\mu)]}{(A+\lambda N)(A+\mu N)-\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}$
only out-degree	$(q-r)\dfrac{[(l-1+\mu)N_{k,l-1}-(l+\mu)N_{kl}]\,[A+\lambda N-(k+\lambda)]}{(A+\lambda N)(A+\mu N)-\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}$
in- and out-degrees	0

Table 3.3: in- and/or out-degree changes occur in the network in case 3

In case 3, the addition of a new directed arc to a node with in-degree k leads to a loss in the number of such nodes. This occurs with rate $(k+\lambda)N_{kl}[\sum_l(l+\mu)-l-\mu]$, which is divided by the denominator. Here $\sum_l(l+\mu)-l-\mu$ represents a node with out-degree equal to l is connected from any other node in the network rather than itself. The denominator shown in the table 3.3 contains two terms. The first term is the total weight $\sum_{k,l}(k+\lambda)(l+\mu)N_{kl}=(A+\lambda N)(A+\mu N)$, and the deducted term is the self loop weights $\sum_{n_{kl}}(k+\lambda)(l+\mu)$.

Now, put all three situations (the changes are listed in tables 3.1, 3.2 and 3.3) together, we get

$$\begin{aligned}
\frac{dN_{kl}}{dt} &= p\,\frac{(k-1+\lambda)N_{k-1}-(k+\lambda)N_k}{A+\lambda N} + p\,\delta_{k0}\delta_{l1} \\
&\quad + q\,\frac{\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}{\sum\limits_{k,l}(k+\lambda)(l+\mu)} \times \\
&\qquad\qquad \times \frac{(k-1+\lambda)(l-1+\mu)N_{k-1,l-1}-(k+\lambda)(l+\mu)N_{kl}}{\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)} \\
&\quad + q\left(1-\frac{\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}{\sum\limits_{k,l}(k+\lambda)(l+\mu)}\right)\times \\
&\qquad\qquad \times \frac{\left[(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}\right]\left[A+\mu N-(l+\mu)\right]}{(A+\lambda N)(A+\mu N)-\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)} \\
&\quad + q\left(1-\frac{\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)}{\sum\limits_{k,l}(k+\lambda)(l+\mu)}\right)\times \\
&\qquad\qquad \times \frac{\left[(l-1+\mu)N_{k,l-1}-(l+\mu)N_{kl}\right]\left[A+\lambda N-(k+\lambda)\right]}{(A+\lambda N)(A+\mu N)-\sum\limits_{n_{kl}}(k+\lambda)(l+\mu)} \\
&= p\,\frac{(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}}{A+\lambda N} + p\,\delta_{k0}\delta_{l1} \\
&\quad + q\,\frac{(k-1+\lambda)(l-1+\mu)N_{k-1,l-1}-(k+\lambda)(l+\mu)N_{kl}}{(A+\lambda N)(A+\mu N)} \\
&\quad + q\,\frac{[(k-1+\lambda)N_{k-1,l}-(k+\lambda)N_{kl}][A+\mu N-(l+\mu)]}{(A+\lambda N)(A+\mu N)} \\
&\quad + q\,\frac{[(l-1+\mu)N_{k,l-1}-(l+\mu)N_{kl}][A+\lambda N-(k+\lambda)]}{(A+\lambda N)(A+\mu N)}
\end{aligned}$$

to complete the proof of the lemma. □

Now we use the above lemma 3.1 in the proof of the following theorem. N_k is defined as the average number of node with in-degree k. N_l is defined as the average number of node with out-degree l.

Theorem 3.2. *Let $k \geq 0, N_k = tN'_k$. As $k \to \infty$ we have $N'_k \sim k^{-\nu_{in}}$, where $\nu_{in} = 2 + p\lambda$. Let $l \geq 0, N_l = tN'_l$. As $l \to \infty$ we have $N'_l \sim l^{-\nu_{out}}$, where $\nu_{out} = 2 + q^{-1} + \mu pq^{-1}$.*

Proof of the Theorem 3.2. To prove the power laws for in- and out-degrees, let us derive the change of the number of nodes with in-degree k

$$\begin{aligned}
\frac{dN_k}{dt} &= \sum_l \frac{dN_{kl}}{dt} \\
&= p\,\frac{(k-1+\lambda)N_{k-1} - (k+\lambda)N_k}{A+\lambda N} + p\,\delta_{k0} \\
&\quad + q\,\frac{(k-1+\lambda)\sum_l (l-1+\mu)N_{k-1,l-1} - (k+\lambda)\sum_l (l+\mu)N_{kl}}{(A+\lambda N)(A+\mu N)} \\
&\quad + q\left(\frac{(k-1+\lambda)\sum_l (A+\mu N - l - \mu)N_{k-1,l}}{(A+\lambda N)(A+\mu N)}\right. \\
&\qquad\qquad \left. - \frac{(k+\lambda)\sum_l (A+\mu N - l - \mu)N_{kl}}{(A+\lambda N)(A+\mu N)}\right) \\
&= p\,\frac{(k-1+\lambda)N_{k-1} - (k+\lambda)N_k}{A+\lambda N} + p\,\delta_{k0} \\
&\quad + q\,\frac{(k-1+\lambda)(A+\mu N)N_{k-1} - (k+\lambda)(A+\mu N)N_k}{(A+\lambda N)(A+\mu N)} \\
&= \frac{(k-1+\lambda)N_{k-1} - (k+\lambda)N_k}{A+\lambda N} + p\,\delta_{k0}.
\end{aligned} \tag{3.8}$$

Following the same assumptions in (Krapivsky et al., 2001), we fill in $A = t$, as well as $N = Ap = tp, N_k = tN'_k$, into Equation (3.8). Then we have,

$$\begin{aligned}
&[t - 1 + \lambda\,(t-1)\,p]\;[t\,N'_k - (t-1)\,N'_k] + (k+\lambda)\,(t-1)\,N'_k \\
&\quad = (k-1+\lambda)\,(t-1)\,N'_{k-1} + p\,\delta_{k0}\,[(t-1) + \lambda\,(t-1)\,p]
\end{aligned} \tag{3.9}$$

After simplifying the left side of Equation (3.9), we get

$$\begin{aligned}
(t-1)\,(1+\lambda\,p)\,N'_k + (k+\lambda)\,(t-1)\,N'_k &= (k-1+\lambda)\,(t-1)\,N'_{k-1} \\
&\quad + p\,\delta_{k0}\,(1+p\,\lambda)\,(t-1)
\end{aligned} \tag{3.10}$$

$$
\begin{aligned}
(1+\lambda\,p+k+\lambda)\;N'_k &= (k-1+\lambda)\;N'_{k-1}+p\,(1+p\,\lambda)\;\delta_{k0}\\
(k+1+(1+p)\;\lambda)\;N'_k &= (k-1+\lambda)\;N'_{k-1}+p\,(1+p\,\lambda)\;\delta_{k0}
\end{aligned}
\tag{3.11}
$$

which has become identical to Equation (6) in paper (Krapivsky et al., 2001).

Similarly we can calculate the change of the number of nodes with the out-degree equal to l

$$
\begin{aligned}
\frac{dN_l}{dt} &= \sum_k \frac{dN_{kl}}{dt}\\
&= p\,\delta_{l1}\\
&\quad + q\,\frac{(l-1+\mu)\sum_k (k-1+\lambda)N_{k-1,l-1}-(l+\mu)\sum_k (k+\lambda)N_{kl}}{(A+\lambda N)(A+\mu N)}\\
&\quad + q\left(\frac{(l-1+\mu)\sum_k (A+\lambda N-k-\lambda)N_{k,l-1}}{(A+\lambda N)(A+\mu N)}\right.\\
&\qquad\qquad \left. -\frac{(l+\mu)\sum_k (A+\lambda N-k-\lambda)N_{kl}}{(A+\lambda N)(A+\mu N)}\right)\\
&= p\,\delta_{l1}+q\,\frac{(l-1+\mu)(A+\lambda N)N_{l-1}-(l+\mu)(A+\lambda N)N_l}{(A+\lambda N)(A+\mu N)}\\
&= q\,\frac{(l-1+\mu)N_{l-1}-(l+\mu)N_l}{A+\mu N}+p\,\delta_{l1}
\end{aligned}
\tag{3.12}
$$

Again, we fill in the assumptions mentioned: $A=t$, as well as $N=Ap=tp, N_l=tN'_l$, into the above equation (3.12). Then we have,

$$
\begin{aligned}
&[(t-1)+\mu\,(t-1)\,p]\;[t\;N'_l-(t-1)\;N'_l]+q\;(l+\mu)\;(t-1)\;N'_l\\
&\qquad = q\;(l-1+\mu)\;(t-1)\;N'_{l-1}+p\,\delta_{l1}\;[(t-1)+\mu\;(t-1)\;p]
\end{aligned}
\tag{3.13}
$$

After simplifying both sides of Equation (3.13), we have

$$
\begin{aligned}
(t-1)\;(1+\mu\,p)\;N'_l+q\;(l+\mu)\;(t-1)\;N'_l &= q\;(l-1+\mu)\;(t-1)\;N'_{l-1}\\
&\quad + p\,\delta_{l1}\;(1+p\,\mu)\;(t-1) \qquad (3.14)\\
(1+\mu\,p+q\,l+q\,\mu)\;N'_l &= q\;(l-1+\mu)\;N'_{l-1}+p\,(1+p\,\mu)\;\delta_{l1}
\end{aligned}
$$

$$\begin{aligned}(q\,l+1+\mu)\,N'_l &= q(l-1+\mu)\,N'_{l-1}+p\,(1+p\mu)\,\delta_{l1}\\ \left(l+\frac{1}{q}+\frac{\mu}{q}\right)\,N'_l &= (l-1+\mu)\,N'_{l-1}+p\,\frac{1+p\,\mu}{q}\,\delta_{l1}\end{aligned} \tag{3.15}$$

which has become identical to Equation (7) in paper (Krapivsky et al., 2001).

Note that although Equation (3.5) is wrong, you get the same result as for the correct Equation (3.1) when you sum all in- (/out-) degrees for the number of nodes with out-degree l (/in-degree k) .

Following the same solution given in Krapivsky et al. (2001), the above recursion Equations (3.11) and (3.15) are solved by the following ratios of gamma functions:

$$N'_k = N'_0\,\frac{\Gamma(k+\lambda)\Gamma(1+(1+p)\lambda+1)}{\Gamma(k+1+(1+p)\lambda+1)\Gamma(\lambda)}, \tag{3.16}$$

with $N'_0 = \dfrac{p(1+p\lambda)}{1+(1+p)\lambda}$, and

$$N'_l = N'_1\,\frac{\Gamma(l+\mu)\Gamma(2+q^{-1}+\mu q^{-1})}{\Gamma(l+1+q^{-1}+\mu q^{-1})\Gamma(1+\mu)}, \tag{3.17}$$

with $N'_1 = \dfrac{p(1+p\mu)}{1+q+\mu}$.

From the asymptotic of the gamma function, the in-degree and out-degree distributions have the power law forms,

$$N'_k \sim k^{-\nu_{in}},\ \nu_{in} = 2+p\lambda, \tag{3.18}$$

$$N'_l \sim l^{-\nu_{out}},\ \nu_{out} = 2+q^{-1}+\mu p q^{-1}. \tag{3.19}$$

to complete the proof of the theorem. □

This KRR algorithm generates a network for a given N, the number of nodes, with the resulting expected number of arcs equals N/p. It introduces multiple arcs and self loops in the process of growing the network. In order to generate a network in the context of web of trust, we consider the following modified versions of KRR algorithm where try to avoid adding multiple arcs and self loops in the process of growing the network.

3.4 The Proposed Algorithms

Having presented the KRR algorithm, in order to generate directed scale-free networks without multiple arcs and self loops, we propose several different algorithms which are the variations of the KRR algorithm. In the KRR algorithm, the possibility of adding a multiple arc or self loop occurs in the process of adding an arc between two existing nodes. Different from the KRR algorithm, the following algorithms take different actions to avoid adding a multiple arc or self loop.

The following pseuducode of algorithm 2 presents algorithms A, B, and C.

Algorithm 2 algorithms A, B, C main()

```
Require: a network $G_0(V_0, E_0)$, $n = |V_0|$, $n \geq 1$
Ensure: $n = N$
 1: repeat
 2:   $a \Leftarrow random[0, 1)$
 3:   if $a < p$ then
 4:     add a new node $m$
 5:     choose node $j$ with a probability that is proportional to weights $w_i$
 6:     generate an arc $m \rightarrow j$
 7:     update node $m$ out-degree and weight: $O_m = O_m + 1$, $v_m = \mu + O_m$
 8:     update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
 9:     increase network size by 1: $n = n + 1$
10:   else
11:     if $E \equiv n(n-1)$ then
12:       go to line 4
13:     else
14:       choose node $i$ with a probability that is proportional to weights $v_i$
15:       choose node $j$ with a probability that is proportional to weights $w_i$
16:       if $i \rightarrow j$ or $i \equiv j$ then
17:         algorithm A: go to line 11
18:         algorithm B: go to line 4
19:         algorithm C: continue
20:       else
21:         generate an arc $i \rightarrow j$
22:         update node $i$ out-degree and weight: $O_i = O_i + 1$, $v_i = \mu + O_i$
23:         update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
24:       end if
25:     end if
26:   end if
27: until there are $N$ nodes: $n = N$
```

3.4.1 Algorithm A

To avoid multiple arcs or self loops being added, when a multiple arc or self loop is chosen, algorithm A redraws the origin node and destination node until this arc is neither multiple arc nor self loop. If the network is fully connected, the algorithm forces a new node to be added in the network. To check whether the network is fully connected or not, we can define a variable A as the total number of arcs in the network. The value of A increases by 1 when a new arc is created. The redraw operation has a worst time complexity $O(N^2)$.

3.4.2 Algorithm B

Different from algorithm A, algorithm B does not redraw the origin node and destination node when a multiple arc or a self loop is chosen. When it is, algorithm B goes to the next iteration with a new node added and attached to the network. Therefore, Algorithm B has a better time complexity than algorithm A.

3.4.3 Algorithm C

In algorithm C, if a multiple arc or self loop is chosen, the algorithm will ignore adding a new arc between two existing nodes and goes to the next iteration. The difference between algorithm C and algorithm B is that when a multiple arc or self loop is chosen, with probability p algorithm C starts the next iteration with a new node. While Algorithm B starts the next iteration with a new node.

Algorithms A, B, and C behave in the same way of adding a new node to the network but take different actions when adding a potential multiple arc or self loop. The algorithms A, B and C all check if the potential arc is a multiple arc or self loop at the time when adding an additional arc between two existing nodes, and take different actions when it is. Also these three algorithms check if the network is fully connected before adding an directed arc between two existing nodes. If that happens, then the algorithms will force a new node to be added into the network. By structure, the algorithms A, B and C are classified as the growth algorithm.

3.4.4 KRRNoLM algorithm

Rather than avoid multiple arcs or self loops being added, we consider to generate a network first by using the KRR algorithm, and then at the end remove all multiple arcs and self loops from the generated network. We call this KRRNoLM (KRR No Loops or Multiple arcs) algorithm. In the KRRNoLM algorithm, the multiple arcs and self loops are removed from the network after the network grows to N nodes. therefore, KRRNoLM is a non-growth algorithm. The process of the KRRNoLM algorithm is presented as Algorithm 3.

Algorithm 3 KRRNoLM algorithm main()

```
Require: a network $G_0(V_0, E_0)$, $n = |V_0|$, $n \geq 1$
Ensure: $n = N$
 1: repeat
 2:     $a \Leftarrow random[0, 1)$
 3:     if  $a < p$ then
 4:         add a new node $m$
 5:         choose node $j$ with a probability that is proportional to weights $w_i$
 6:         generate an arc $m \to j$
 7:         update node $m$ out-degree and weight: $O_m = O_m + 1$, $v_m = \mu + O_m$
 8:         update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
 9:         increase network size by 1: $n = n + 1$
10:     else
11:         choose node $i$ with a probability that is proportional to weights $v_i$
12:         choose node $j$ with a probability that is proportional to weights $w_i$
13:         generate an arc $i \to j$
14:         update node $i$ out-degree and weight: $O_i = O_i + 1$, $v_i = \mu + O_i$
15:         update node $j$ in-degree and weight: $I_j = I_j + 1$, $w_j = \lambda + I_j$
16:     end if
17: until there are $N$ nodes: $n = N$
18: remove all multiple arcs and self loops
```

Compared to the KRR algorithm, a drawback of the KRRNoLM algorithm is that it generates networks with an unpredictable number of arcs, since it is unknown how many arcs are multiple arcs or self loops.

Before moving to the next section, let us compare the network properties generated by all algorithms and the order of all algorithms for CPU time complexity in Table 3.4. In Table 3.4 for the network properties we concern the number of nodes, the number of arcs, the estimated power law exponent values for in-degree and out-degree, respectively. We also concern the pure growth feature. The pure growth here means that whether an algorithm generates the resulting

Algorithm	Network Properties				Com-	Pure
	#nodes	#arcs	α_{in}	α_{out}	plexity	growth
KRR	N	$\frac{N}{p}$	$2 + p\lambda$	$2 + \dfrac{p(\mu + 1)}{1 - p}$	$O(N)$	yes
A	N	unknown	unknown	unknown	$O(N^3)$	yes
B	N	unknown	unknown	unknown	$O(N)$	yes
C	N	unknown	unknown	unknown	$O(N)$	yes
KRRNoLM	N	unknown	unknown	unknown	$O(N)$	no

Table 3.4: Network properties and order of CPU use for various algorithms

growth networks or not. This feature is of importance, as growing the size of the network in a manner that intuitively resembles real networks. We can see from Table 3.4 in the first four columns that only the KRR algorithm can generate a network with the expected number of arcs equals $\frac{N}{p}$. For all other algorithms the number of arcs is not available. For algorithms A, B, and C, the probability of choosing a multiple arc or self loop is unpredictable. For the KRRNoLM algorithm, the number of multiple arcs or self loops is unknown. It causes the scale free exponents difficult to determine. In term of pure growth feature, the KRR algorithm and algorithms A, B, and C are pure growth. The KRRNoLM algorithm does not hold, as all multiple arcs and self loops are removed from the generated network at the end. We also see from the table that algorithm A has a worst time complexity which is caused by the redraw operation. Note that the CPU time complexity does not include aspects that are required in all algorithms, such as updates of in-degrees and out-degrees.

Typically the judgement of scale-free networks is based on visualisations. If the degree distribution looks roughly a straight line on a log-log plot used in the figure, one claims the network follow a power law. Here we plot the node in-degree and out-degree distributions of networks generated by various algorithms in a log-log figure. We consider all the algorithms with the same parameter values used in paper (Krapivsky et al., 2001) and network size 100,000 nodes. For each algorithm, we consider 100 network samples to be generated. Figures 3.2 and 3.3 present the average node in-degree and out-degree distributions of

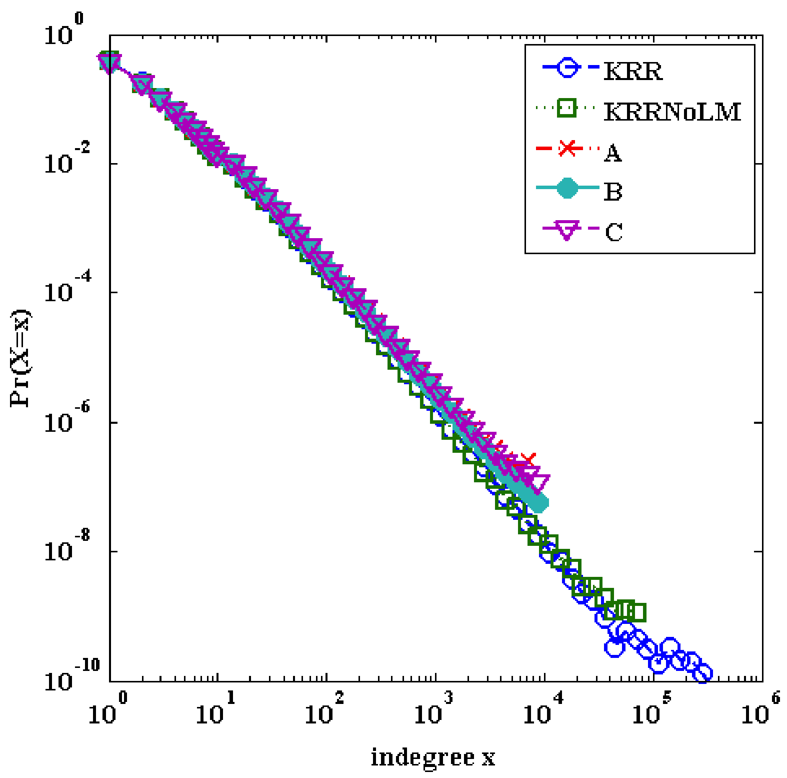

Figure 3.2: The average in-degree node distribution of networks generated by various algorithms, versus the in-degree of the networks with 100, 000 nodes. The data sets are plotted using logarithmic binning.

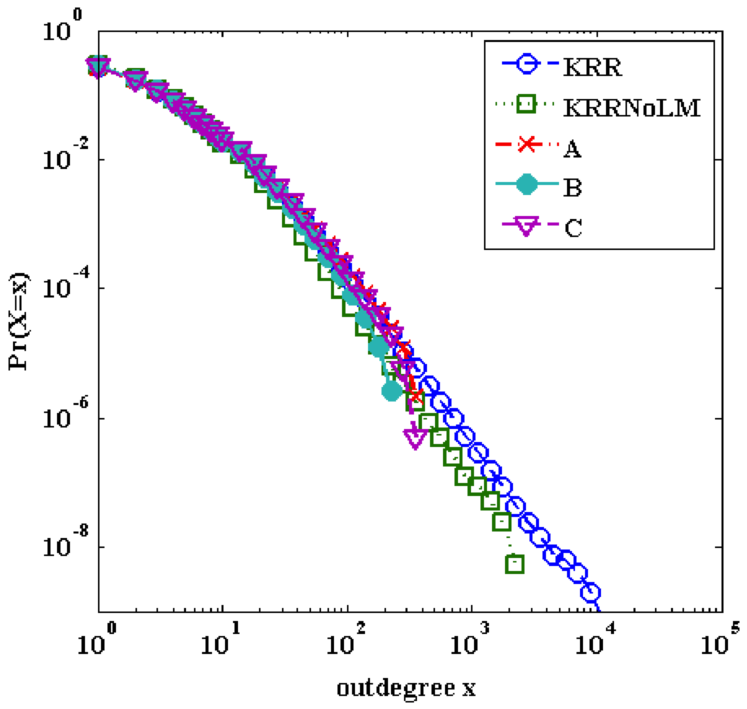

Figure 3.3: The average out-degree node distribution of networks generated by various algorithms, versus the out-degree of the networks with 100, 000 nodes. The data sets are plotted using logarithmic binning.

the networks generated by various algorithms, respectively. It can be observed that both the in-degree and out-degree distributions of each algorithm look roughly straight on a log-log plot. The KRR algorithm has a few nodes with large in-degree/out-degree.

3.5 A Mathematical Proof of Scale-Freeness of Algorithms

In this section, we provide a mathematical framework to prove under which conditions the algorithm can generate networks with the scale-free feature. Let us recap the process of a growth network generated by the KRR algorithm. A directed network grows by adding single arc at discrete time steps. At time t the network G^t has A^t arcs, and N^t nodes. From G^t to G^{t+1}, with probability p the algorithm adds a new node together with an arc from the new node to an existing node (called operation (i)), while with probability $q = 1 - p$ an additional arc is added between existing nodes (called operation (ii)). The arc to be added to the network depends on the weights of the origin and the destination nodes.

We start with the work on the node's in-degree distribution. A list of definitions is needed as follows.

$$
\begin{aligned}
t &= \text{a particular time step. At each time step, an arc is added} \\
G^t &= \text{the growth network at a particular time } t \\
A^t &= \text{the total number of arcs in } G^t \\
N^t &= \text{the total number of nodes in } G^t \\
N^t_{\text{in\_}i} &= \text{the number of nodes with in-degree } i \text{ in } G^t \\
N^t_{\text{out\_}j} &= \text{the number of nodes with out-degree } j \text{ in } G^t \\
p^t_{\text{in\_}i,\text{out\_}j} &: \text{the probability that an arc is added from a particular node} \\
&\quad\ \text{with out-degree } j \text{ to a particular node with in-degree } i\text{, given} \\
&\quad\ \text{an arc is added at } t. \\
p^t_{\text{in\_}i} &: \text{the probability that an arc is added from any node to a part-} \\
&\quad\ \text{icular node with in-degree } i\text{, given an arc is added at } t. \\
p^t_{\text{in\_}i} &= \sum_{\text{out\_}j} p^t_{\text{in\_}i,\text{out\_}j}
\end{aligned}
$$

$$
\begin{aligned}
p^t_{\text{out\_}j} &: \text{the probability that an arc is added from a particular node} \\
&\quad \text{with out-degree } j \text{ to any node, given an arc is added at } t. \\
p^t_{\text{out\_}j} &= \sum_{\text{in\_}i} p^t_{\text{in\_}i,\text{out\_}j} \\
N^t_{\text{in\_}i-1} p^t_{\text{in\_}i-1} &= \text{Prob}\{\text{one of nodes with in-degree } i-1 \text{ in } G^t \text{ becomes one} \\
&\qquad \text{with in-degree } i \text{ in } G^{t+1} \mid \text{an arc is added in } G^t\} \\
N^t_{\text{in\_}i} p^t_{\text{in\_}i} &= \text{Prob}\{\text{one of nodes with in-degree } i \text{ in } G^t \text{ becomes one} \\
&\qquad \text{with in-degree } i+1 \text{ in } G^{t+1} \mid \text{an arc is added in } G^t\} \\
N^t_{\text{out\_}j-1} p^t_{\text{out\_}j-1} &= \text{Prob}\{\text{one of nodes with out-degree } j-1 \text{ in } G^t \text{ becomes one} \\
&\qquad \text{with out-degree } j \text{ in } G^{t+1} \mid \text{an arc is added in } G^t\} \\
N^t_{\text{out\_}j} p^t_{\text{out\_}j} &= \text{Prob}\{\text{one of nodes with out-degree } j \text{ in } G^t \text{ becomes one} \\
&\qquad \text{with out-degree } j+1 \text{ in } G^{t+1} \mid \text{an arc is added in } G^t\}
\end{aligned}
$$

Giving p+q=1, as well as $\sum_{\text{in\_}i} p_{\text{in\_}i} = 1$, $\sum_{\text{out\_}j} p_{\text{out\_}j} = 1$, $A^t = t$, $N^t = pt$, $N^t_{\text{in\_}i} = tN'_{\text{in\_}i}$, $N^t_{\text{out\_}j} = tN'_{\text{out\_}j}$, Now we consider how the number of nodes with in-degree i in G^t changes as t increases by 1. Let G^t be given and in going from G^t to G^{t+1}, the changes of the number of nodes with in-degree i could have come from two cases: either a node with in-degree $i-1$ at time t had a new arc attached to it at time $t+1$, or a node with in-degree i at time t had a new arc attached to it at time $t+1$. Given that from G^t to G^{t+1} we perform operation (i) or (ii), the probability that a particular node of in-degree i has its in-degree increased is $p^t_{\text{in\_}i}$. Since G^t has the exactly $N^t_{\text{in\_}i}$ nodes of in-degree i, the chance that one of these becomes a node of in-degree $i+1$ in G^{t+1} is exactly $(p+q)N^t_{\text{in\_}i}p^t_{\text{in\_}i}$. With probability $(p+q)N^t_{\text{in\_}i-1}p^t_{\text{in\_}i-1}$ a node of in-degree $i-1$ in G^t becomes a node of in-degree i in G^{t+1}. Putting these effects together, the expected value of $N^{t+1}_{\text{in\_}i}$ in G^{t+1}, which is defined as $E(N^{t+1}_{\text{in\_}i}|G^t)$, is

$$
\begin{aligned}
E(N^{t+1}_{\text{in\_}i}|G^t) &= N^t_{\text{in\_}i} + (p+q)(N^t_{\text{in\_}i-1}p^t_{\text{in\_}i-1} - N^t_{\text{in\_}i}p^t_{\text{in\_}i}) + p1_{\{\text{in\_}i=0\}} \\
&= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}p^t_{\text{in\_}i-1} - N^t_{\text{in\_}i}p^t_{\text{in\_}i} + p1_{\{\text{in\_}i=0\}} \qquad (3.20)
\end{aligned}
$$

where $i \geq 0$ and we write 1_A for the indicator function which is 1 if the event A holds and 0 otherwise. $p1_{\{\text{in\_}i=0\}}$ means that a new node with in-degree 0 is added to the network from time t to time $t+1$.

Similarly, for out-degrees, we consider for each j how $N^t_{\text{out\_}j}$ nodes of out-degree

j in G^t changes as time t increases by 1. Given G^t, with probability p a new node with out-degree 1 is added to the network at the next step, and with probability $q = 1 - p$ the out-degree of an old node is increased. Given that from G^t to G^{t+1} we perform operation (ii), the probability that a particular node of out-degree j has its out-degree increased is $p^t_{\text{out\_}j}$. Since G^t has the exactly $N^t_{\text{out\_}j}$ nodes of out-degree j, the chance that one of these becomes a node of out-degree $j+1$ in G^{t+1} is exactly $qN^t_{\text{out\_}j}p^t_{\text{out\_}j}$. With probability $qN^t_{\text{out\_}j-1}p^t_{\text{out\_}j-1}$ a node of out-degree $j-1$ in G^t becomes a node of out-degree j in G^{t+1}. Putting these effects together, the expected value of $N^{t+1}_{\text{out\_}j}$ in G^{t+1}, which is defined as $E(N^{t+1}_{\text{out\_}j}|G^t)$, is

$$E(N^{t+1}_{\text{out\_}j}|G^t) = N^t_{\text{out\_}j} + q(N^t_{\text{out\_}j-1}p_{\text{out\_}j-1} - N^t_{\text{out\_}j}p_{\text{out\_}j}) + p1_{\{\text{out\_}j=1\}}, \tag{3.21}$$

where $j \geq 1$ and $p1_{\{\text{out\_}j=1\}}$ means that a new node with out-degree 1 is added to the network at time $t+1$.

Given the above definitions, now we apply it to the KRR algorithm and algorithm A respectively, to show the conditions under which growth algorithms are power law.

3.5.1 KRR algorithm

In KRR the arc to be added to the network depends on the weights of the origin and the destination nodes. These weights $w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}$ are defined as follows:

$$w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j} = (i+\lambda)(j+\mu) \tag{3.22}$$

where the weights $i+\lambda, j+\mu$ are already introduced in Equation (3.4).

Theorem 3.3. *Let $i \geq 0$. As $i \to \infty$ we have $N'_{in\_i} \sim i^{-\nu_{in}}$. Let $j \geq 0$. As $j \to \infty$ we have $N'_{out\_j} \sim j^{-\nu_{out}}$.*

Proof of the Theorem 3.3. The probability $p^{\text{KRR}}_{\text{in\_i}}$, that, given an arc is added to the network, it is connected to a node with in-degree i is then given by

$$p^{\text{KRR}}_{in\_i} = \sum_{\text{out\_}j} p^{\text{KRR}}_{\text{in\_}i,\text{out\_}j} = \sum_{\text{out\_}j} \frac{w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}}{\sum w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}} \tag{3.23}$$

therefore,

$$\begin{aligned}
E(N^{t+1}_{\text{in\_}i}|G^t) &= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}p^{\text{KRR}}_{\text{in\_}i-1} - N^t_{\text{in\_}i}p^{\text{KRR}}_{\text{in\_}i} + p1_{\{\text{in\_}i=0\}} \\
&= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}\sum_{\text{out\_}j} p^{\text{KRR}}_{\text{in\_}i-1,\text{out\_}j} - N^t_{\text{in\_}i}\sum_{\text{out\_}j} p^{\text{KRR}}_{\text{in\_}i,\text{out\_}j} + p1_{\{\text{in\_}i=0\}} \\
&= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}\sum_{\text{out\_}j} \frac{w^{\text{KRR}}_{\text{in\_}i-1,\text{out\_}j}}{\sum w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}} - N^t_{\text{in\_}i}\sum_{\text{out\_}j} \frac{w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}}{\sum w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}} \\
&\quad + p1_{\{\text{in\_}i=0\}} \\
&= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}\frac{\sum_{\text{out\_}j} w^{\text{KRR}}_{\text{in\_}i-1,\text{out\_}j}}{\sum w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}} - N^t_{\text{in\_}i}\frac{\sum_{\text{out\_}j} w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}}{\sum w^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}} \\
&\quad + p1_{\{\text{in\_}i=0\}} \\
&= N^t_{\text{in\_}i} + N^t_{\text{in\_}i-1}\frac{i-1+\lambda}{A^t+\lambda N^t} - N^t_{\text{in\_}i}\frac{i+\lambda}{A^t+\lambda N^t} + p1_{\{\text{in\_}i=0\}} \qquad (3.24)
\end{aligned}$$

Assuming the node in-degree of the KRR algorithm is power law distributed, then $N'_{\text{in\_}i} \sim i^{v_{\text{in}}}$. KRR algorithm satisfies $N^t_{\text{in\_}i} = tN'_{\text{in\_}i}$, $A^t = t$, $N^t = pt$, then fill them in and we have

$$\begin{aligned}
(t+1)N'_{\text{in\_}i} &= tN'_{\text{in\_}i} + tN'_{\text{in\_}i-1}\frac{i-1+\lambda}{A^t+\lambda N^t} - tN'_{\text{in\_}i}\frac{i+\lambda}{A^t+\lambda N^t} + p1_{\{\text{in\_}i=0\}} \Rightarrow \\
N'_{\text{in\_}i} &= tN'_{\text{in\_}i-1}\frac{i-1+\lambda}{t+\lambda pt} - tN'_{\text{in\_}i}\frac{i+\lambda}{t+\lambda pt} + p1_{\{\text{in\_}i=0\}} \Rightarrow \\
i^{v_{\text{in}}} &= (i-1)^{v_{\text{in}}}\frac{i-1+\lambda}{1+\lambda p} - i^{v_{\text{in}}}\frac{i+\lambda}{1+\lambda p} + p1_{\{\text{in\_}i=0\}} \Rightarrow \\
(1+\lambda p+i+\lambda)i^{v_{\text{in}}} &= (i-1+\lambda)(i-1)^{v_{\text{in}}} + p(1+\lambda p)1_{\{\text{in\_}i=0\}} \qquad (3.25)
\end{aligned}$$

Since $i \geq 1$, then take logarithm in both sides of Equation (3.25), and we obtain

$$\log(1+\lambda p+i+\lambda) + v_{\text{in}}\log i = \log(i-1+\lambda) + v_{\text{in}}\log(i-1) \qquad (3.26)$$

This equality is true if $i \to \infty$. In other words, $N'_{\text{in\_}i}$ is power law if $i \to \infty$.

Now we consider the expected value of the number of nodes with out-degree j,

$$\begin{aligned}
E(N^{t+1}_{\text{out\_}j}|G^t) &= N^t_{\text{out\_}j} + q(N^t_{\text{out\_}j-1}p^{\text{KRR}}_{\text{out\_}j-1} - N^t_{\text{out\_}j}p^{\text{KRR}}_{\text{out\_}j}) + p1_{\{\text{out\_}j=1\}} \\
&= N^t_{\text{out\_}j} + (1-p)\left(N^t_{\text{out\_}j-1}\sum_{\text{in\_}i} p^{\text{KRR}}_{\text{in\_}i,\text{out\_}j-1} - N^t_{\text{out\_}j}\sum_{\text{in\_}i} p^{\text{KRR}}_{\text{in\_}i,\text{out\_}j}\right) \\
&\quad + p1_{\{\text{in\_}i=1\}}
\end{aligned}$$

$$
\begin{aligned}
&= N^t_{\mathrm{out}\_j} + (1-p)\Bigg(N^t_{\mathrm{out}\_j-1} \sum_{\mathrm{in}\_i} \frac{w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j-1}}{\sum w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}} \\
&\qquad\qquad - N^t_{\mathrm{out}\_j} \sum_{\mathrm{in}\_i} \frac{w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}}{\sum w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}} \Bigg) \\
&\quad + p1_{\{\mathrm{out}\_i=1\}} \\
&= N^t_{\mathrm{out}\_j} + (1-p)\Bigg(N^t_{\mathrm{out}\_j-1} \frac{\sum_{\mathrm{in}\_i} w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j-1}}{\sum w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}} \\
&\qquad\qquad - N^t_{\mathrm{out}\_j} \frac{\sum_{\mathrm{in}\_i} w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}}{\sum w^{\mathrm{KRR}}_{\mathrm{in}\_i,\mathrm{out}\_j}} \Bigg) \\
&\quad + p1_{\{\mathrm{out}\_i=1\}} \\
&= N^t_{\mathrm{out}\_j} + (1-p)\left(N^t_{\mathrm{out}\_j-1} \frac{j-1+\mu}{A^t+\mu N^t} - N^t_{\mathrm{out}\_j} \frac{j+\mu}{A^t+\mu N^t} \right) \\
&\quad + p1_{\{\mathrm{out}\_i=1\}} \qquad (3.27)
\end{aligned}
$$

Assuming the node out-degree of the KRR algorithm is power law distributed, then $N'_{\mathrm{out}\_j} \sim j^{v_{\mathrm{out}}}$. KRR algorithm satisfies $N^t_{\mathrm{out}\_j} = tN'_{\mathrm{out}\_j}$, $A^t = t$, $N^t = pt$, then fill them in and we have

$$
\begin{aligned}
(t+1)N'_{\mathrm{out}\_j} &= tN'_{\mathrm{out}\_j} + (1-p)\Bigg(tN'_{\mathrm{out}\_j-1} \frac{j-1+\mu}{A^t+\mu N^t} \\
&\qquad - tN'_{\mathrm{out}\_j} \frac{j+\mu}{A^t+\mu N^t} \Bigg) \\
&\quad + p1_{\{\mathrm{out}\_j=1\}} \Rightarrow \\
N'_{\mathrm{out}\_j} &= (1-p)\left(tN'_{\mathrm{out}\_j-1} \frac{j-1+\mu}{t+\mu pt} - tN'_{\mathrm{out}\_j} \frac{j+\mu}{t+\mu pt} \right) \\
&\quad + p1_{\{\mathrm{out}\_j=1\}} \Rightarrow \\
j^{v_{\mathrm{out}}} &= (1-p)\left((j-1)^{v_{\mathrm{out}}} \frac{j-1+\mu}{1+\mu p} - j^{v_{\mathrm{out}}} \frac{j+\mu}{1+\mu p} \right) \\
&\quad + p1_{\{\mathrm{out}\_j=1\}} \Rightarrow \\
(1+\mu+(1-p)j)j^{v_{\mathrm{out}}} &= (1-p)(j-1+\mu)(j-1)^{v_{\mathrm{out}}} + p(1+\mu p)1_{\{\mathrm{out}\_j=1\}} \qquad (3.28)
\end{aligned}
$$

Since $j \geq 1$, then take logarithm in both sides of the equation, and we obtain

$$
\log(1+\mu+(1-p)j) + v_{\mathrm{out}} \log j = \log((1-p)(j-1+\mu)) + v_{\mathrm{out}} \log(j-1) \qquad (3.29)
$$

This equality is true if $j \to \infty$. In other words, N'_{out_j} is power law if $j \to \infty$. $\square$

3.5.2 Algorithm A

Now we move on to algorithm A. In algorithm A, not allowing multiple arcs or self loops, weights are defined as follows:

$$w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j}) = \begin{cases} (i+\lambda)(j+\mu) & \text{if} \quad x_{\mathrm{out}\_j} \not\to y_{\mathrm{in}\_i} \\ 0 & \text{if} \quad x_{\mathrm{out}\_j} \to y_{\mathrm{in}\_i} \text{ or } \ x = y \end{cases}$$

The main difference on weights from the KRR algorithm is that $w_{i,j}$ is set to 0 in algorithm A if an arc already exists or if it is a loop. The probability $p(y_{\mathrm{in}\_i})^{\mathrm{A}}$, that, given an arc is added to the network, it is connected to a node y with in-degree i is then given by

$$p^{\mathrm{A}}(y_{\mathrm{in}\_i}) = \sum_{x_{\mathrm{out}\_j}} p^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j}) = \sum_{x_{\mathrm{out}\_j}} \frac{w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})}{\sum w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})} \tag{3.30}$$

therefore,

$$\begin{aligned} E(N^{t+1}_{\mathrm{in}\_i}|G^t) &= N^t_{\mathrm{in}\_i} + N^t_{\mathrm{in}\_i-1} p^{\mathrm{A}}(y_{\mathrm{in}\_i-1}) - N^t_{\mathrm{in}\_i} p^{\mathrm{A}}(y_{\mathrm{in}\_i}) + p1_{\{\mathrm{in}\_i=0\}} \\ &= N^t_{\mathrm{in}\_i} + N^t_{\mathrm{in}\_i-1} \sum_{\mathrm{out}\_j} p^{\mathrm{A}}(y_{\mathrm{in}\_i-1}, x_{\mathrm{out}\_j}) - N^t_{\mathrm{in}\_i} \sum_{\mathrm{out}\_j} p^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j}) \\ &\qquad + p1_{\{\mathrm{in}\_i=0\}} \\ &= N^t_{\mathrm{in}\_i} + N^t_{\mathrm{in}\_i-1} \sum_{\mathrm{out}\_j} \frac{w^{\mathrm{A}}(y_{\mathrm{in}\_i-1}, x_{\mathrm{out}\_j})}{\sum w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})} \\ &\qquad - N^t_{\mathrm{in}\_i} \sum_{\mathrm{out}\_j} \frac{w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})}{\sum w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})} + p1_{\{\mathrm{in}\_i=0\}} \\ &= N^t_{\mathrm{in}\_i} + N^t_{\mathrm{in}\_i-1} \frac{\sum_{\mathrm{out}\_j} w^{\mathrm{A}}(y_{\mathrm{in}\_i-1}, x_{\mathrm{out}\_j})}{\sum w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})} \\ &\qquad - N^t_{\mathrm{in}\_i} \frac{\sum_{\mathrm{out}\_j} w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})}{\sum w^{\mathrm{A}}(y_{\mathrm{in}\_i}, x_{\mathrm{out}\_j})} + p1_{\{\mathrm{in}\_i=0\}} \\ &= N^t_{\mathrm{in}\_i} + N^t_{\mathrm{in}\_i-1} \times \\ &\qquad \times \frac{(i-1+\lambda)\left(A^t + \mu N^t - \sum_{x,y|x\to y \bigvee x=y}(j+\mu)\right)}{(A^t+\lambda N^t)(A^t+\mu N^t) - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)(j+\mu)} \\ &\qquad - N^t_{\mathrm{in}\_i} \frac{(i+\lambda)\left(A^t + \mu N^t - \sum_{x,y|x\to y \bigvee x=y}(j+\mu)\right)}{(A^t+\lambda N^t)(A^t+\mu N^t) - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)(j+\mu)} \\ &\qquad + p1_{\{\mathrm{in}\_i=0\}} \end{aligned} \tag{3.31}$$

where, the probability $\frac{\sum_{x,y|x\to y \bigvee x=y}((i+\lambda)(j+\mu))}{\sum w^A(y_{\text{in\_}i}, x_{\text{out\_}j})}$ that, given an arc is chosen, it is a multiple arc or self loop is unpredictable, so it is very difficult to simplify the equation. If we know this probability, then we know if Algorithm A holds a power law form on the in-degree distribution. Similarly for the out-degrees,

$$\begin{aligned}
E(N^{t+1}_{\text{out\_}j}|G^t) &= N^t_{\text{out\_}j} + q\left(N^t_{\text{out\_}j-1}p^{\text{A}}(y_{\text{out\_}j-1}) - N^t_{\text{out\_}j}p^{\text{A}}(y_{\text{out\_}j})\right) + p1_{\{\text{out\_}j=1\}} \\
&= N^t_{\text{out\_}j} + q\left(N^t_{\text{out\_}j-1}\sum_{\text{in\_}i} p^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j-1}) \right.\\
&\qquad\qquad \left. - N^t_{\text{out\_}j}\sum_{\text{in\_}i} p^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})\right) \\
&\qquad + p1_{\{\text{out\_}j=1\}} \\
&= N^t_{\text{out\_}j} + qN^t_{\text{out\_}j-1}\sum_{\text{in\_}i}\frac{w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j-1})}{\sum w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})} \\
&\qquad - qN^t_{\text{out\_}j}\sum_{\text{in\_}i}\frac{w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})}{\sum w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})} \\
&\qquad + p1_{\{\text{out\_}j=1\}} \\
&= N^t_{\text{out\_}j} + qN^t_{\text{out\_}j-1}\frac{\sum_{\text{in\_}i} w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j-1})}{\sum w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})} \\
&\qquad - qN^t_{\text{out\_}j}\frac{\sum_{\text{in\_}i} w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})}{\sum w^{\text{A}}(y_{\text{in\_}i}, x_{\text{out\_}j})} \\
&\qquad + p1_{\{\text{out\_}j=1\}} \\
&= N^t_{\text{out\_}j} \\
&\quad + qN^t_{\text{out\_}j-1}\times \\
&\qquad \times\frac{(j-1+\mu)\left(A^t+\lambda N^t - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)\right)}{(A^t+\lambda N^t)(A^t+\mu N^t) - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)(j+\mu)} \\
&\quad - qN^t_{\text{out\_}j}\frac{(j+\mu)\left(A^t+\lambda N^t - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)\right)}{(A^t+\lambda N^t)(A^t+\mu N^t) - \sum_{x,y|x\to y \bigvee x=y}(i+\lambda)(j+\mu)} \\
&\quad + p1_{\{\text{out\_}j=1\}} \qquad (3.32)
\end{aligned}$$

Again, if we know the probability that given an arc is chosen, it is a multiple arc or self loop, then we know if algorithm A holds a power law form on the out-degree distribution.

3.6 Summary

In this chapter, we proposed several algorithms in order to generate directed scale free networks without multiple arcs or self loops. All the algorithms which modify an existing network generating algorithm can be classified in two groups: the growth (Algorithms A, B and C) and non-growth (the KRRNoLM Algorithm) algorithms. A corrected mathematical proof is presented to prove the power law distribution of the generated networks using an existing network generating algorithm. Also a mathematical framework is presented to prove under which conditions the algorithm can generate networks with the scale-free feature. The mathematical generic framework shows that the growth algorithms are power law when the size of the networks as well as in-degree and out-degree go to infinity. In the next chapter we will evaluate if these algorithms generate scale-free networks using statistical tests.

Chapter 4

Statistical Test for Power Law Distribution Verification

The previous chapter presented a mathematical proof for the power law node degree distribution of the generated network. The power law feature holds under certain conditions, in particular, the size of the network, as well as in-degree and out-degree go to infinity. Different from the theory, all real-life networks are finite. Therefore, in this chapter we employ statistical tests on the experimental data generated by each algorithm we mentioned in the previous chapter.

In the first part, in Section 4.1 we introduce the principled statistical framework done by Clauset et al. (2009), and explain how it quantifies power-law behavior in a particular data set. To demonstrate its use, in the Section 4.2, we apply the statistical framework to some synthetic data sets drawn from the real power law distribution, and analyse the test results. Section 4.3 presents the test results of the experimental data generated by all the algorithms introduced in the previous chapter. Lastly, we discuss the test results of the empirical data of OpenPGP Web of Trust network in Section 4.4.

We base our work on (Clauset et al., 2009), in which power law distribution verification is carried out using a statistical analysis framework, and name this statistical framework as CSN statistical framework by the authors' names. The details of the statistical analysis process are as follows.

4.1 CSN Statistical Framework

Suppose we have an original data set, which has m elements, $A = \{x_i : i = 1, \ldots, m\}$. Assuming this original data set follows a power law distribution, the CSN framework first estimates the two parameters x_{min} and α such that:

$$p(x) = Pr(X = x) = x^{-\alpha} \quad \text{for } x \geq x_{min} \cdot$$

Note that the original data set would typically be real network data, but here for algorithms we introduced, and then the CSN framework determines whether this power law hypothesis is plausible by quantitative testing. The approach is to generate many synthetic data sets from the same power law distribution, and measure how closely these synthetic data sets match the power law form (the theoretical distribution) individually, and compare the measure results with similar measurement on the original data. We write the synthetic data sets as:

$$C = \{C_k : k = 1, \ldots, n\},\ C_k = \{z_i : i = 1, \ldots, m\},$$

where z_i are the individual data points of the sample, m is as above (the size of the data set) and n is a variable set in the test (e.q. $n = 600$ in our study).

The difference with the cases in (Clauset et al., 2009) is that here we can generate as many data sets represent the "original data" as needed (whereas for real data, there typically is only one data set). Note that each synthetic data set has the same size m of the original data set. An algorithm to generate the synthetic data can be found in (Clauset et al., 2009).

To measure the difference between two probability distributions, particularly in use of the analysis of power law distributions, the CSN framework suggests that the Kolmogorov-Smirnov statistic test (K-S test) gives excellent results. We explain the process detail by using an example of measuring the original data set A as above. Let B be a subset of the original data set A, written as:

$$B = \{x_j : x_j \in A \text{ and } x_j \geq x_{min}\},$$

where the individual data value not smaller than x_{min}. Denote $S(x)$ as the cumulative distribution function (CDF) of B:

$$S(x) = \frac{|\{y : y \leq x \text{ and } y \in B\}|}{|B|},$$

and $P(x)$ is the CDF of the hypothetic power law model $p(x)$ as above,

$$P(x) = Pr(X \leq x) = \sum_{x'=x_{min}}^{x} p(x').$$

Then the measure for the discrepancy between two CDFs:

$$D_A = \max_{x \geq x_{min}} |S(x) - P(x)|.$$

Similarly, the measurement applies on synthetic data sets and we name D_{C_k} for each. Finally, the CSN framework calculates a p_{test} value, which is the fraction of the number of synthetic data sets whose measurements on discrepancy are larger than the original one.

$$p_{test} = \frac{|\{C_k : D_{C_k} \geq D_A\}|}{n} \tag{4.1}$$

To obtain an accurate estimate of p_{test}, the value of n, the number of synthetic data sets to generate needs to be considered. The authors in (Clauset et al., 2009) point out that at least $\frac{1}{4}\epsilon^{-2}$ (ϵ is the variance of the estimated mean value of p_{test}) synthetic data sets are needed if one wishes the p_{test} to be accurate to within about ϵ of the true value. In our study, we generate 600 synthetic sets to make sure our p_{test} to be accurate to about 0.02.

Once p_{test} is calculated, one can make a decision about whether reject the hypothesis. The authors in (Clauset et al., 2009) made a conservative choice and set the critical value of p_{test} to 0.1. If p_{test} is smaller than or equals 0.1, the power law distribution hypothesis is rejected, because there is a probability of 1 in 10 or less that synthetic data set more poorly matches the power law model than the original data set.

4.2 Synthetic dataset test

Using the statistical framework outlined above, we first conduct statistical tests on some synthetic data sets, which are generated from a true power law model via setting α value. It is mainly to illustrate to the reader the statistical test results of the data sets drawn from the real power law distribution. And it can also be a reference to which we can compare to datasets generated from the previous introduced algorithms.

In the tests, we consider the value for α in a range of $[2, 3]$ as per previously mentioned findings. In this case, the expected number of arcs of a node is finite and the variance is infinite. Considering sample-to-sample variation, for each α value we generate 100 synthetic data set samples. In each synthetic data set we consider minimum degree value $x_{min} = 1$, the maximum degree value $x_{max} = 20000$, and the sub-sample size 2500. The synthetic data sets can be denoted mathematically as $S = \{S_k : k = 1, \ldots, 100\}, S_k = \{x_i : 1 \leq x_i \leq 20000\}, |S_k| = 2500$. We set x_{max} value high enough to get the expected size of S_k. The average statistical test result is derived:

$$\overline{p_{test}} = \frac{\sum\limits_{k=1}^{100} p_{test}(S_k)}{100},$$

where $p_{test}(S_k)$ are the statistical test result for individual synthetic data set S_k.

The function to generate synthetic data sets is implemented in technical computing software MATLAB_$R2009b$. All the synthetic data sets are sampled from a power law model. Due to the random nature of the sampling process, there will be always some deviations between synthetic data sets and the power law form. A synthetic data sample is therefore not guaranteed to pass the test. When significant number of synthetic data tests are taken, the value of p_{test} should converge on the theoretical value of 0.5, demonstrating the power law hypothesis is reasonable. When we fit the correct model to the data sets, the resulting p_{test} is uniformly distributed, and the theoretical value of 0.5 is expected.

Figure 4.1 presents the synthetic data test results with different α values. In Figure 4.1, we can see that the average p_{test} values only slightly deviate from 0.5,

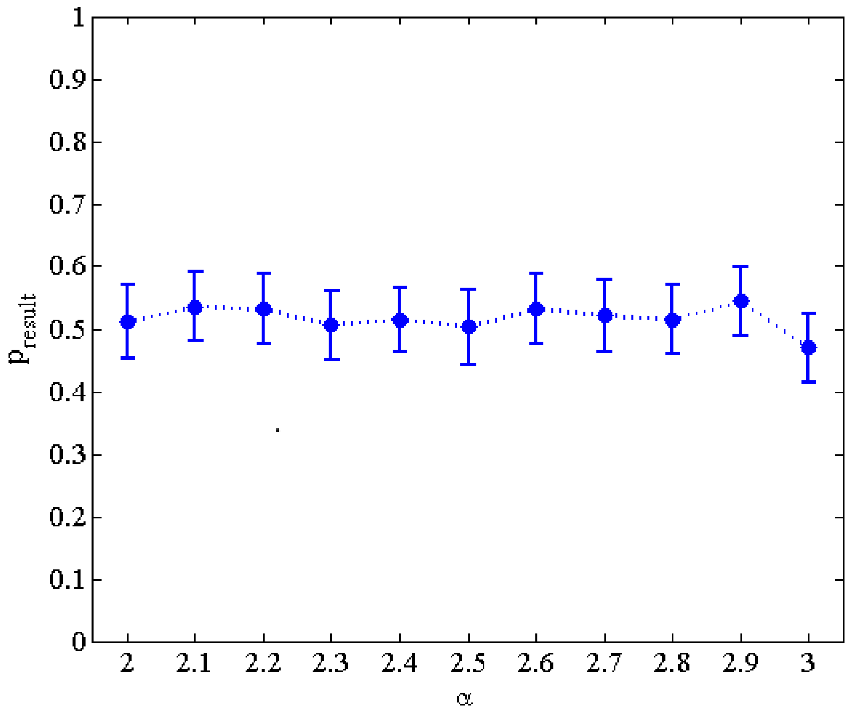

Figure 4.1: The statistical test results of synthetic data sets drawn from a power law model $\Pr[X = x] \propto x^{-\alpha}$, $x_{min} = 1$

which implies that all the synthetic data sets with α value between 2 and 3 pass the test and the samples size we considered is big enough.

Next we will have a look at the results which are derived from the statistical analysis for these algorithms.

4.3 Experimental Results

In this section, we will look at the statistical test results on the various networks which are generated by using the algorithms discussed in Section 3.3 and Section 3.4.

All the network generating algorithms are implemented and executed within the Java Peersim simulation environment for peer-to-peer networks (Jesi, 2004). Details of how to generate the network by using these algorithms are presented in Section 5.3.4. By running the simulations, the generated network data set samples are used as an input of CSN framework in MATLAB code, which are available from the website (Clauset, n.d.). We use MATLAB for performing the data calculations and plotting of graphs.

We first look at the node in-degree and out-degree distributions of networks generated by using these algorithms in a log-log figure. We consider all the algorithms with the same parameter values used in paper (Krapivsky et al., 2001) $p = 0.1333$, $\lambda = 0.75$, $\mu = 3.55$, as representative for web hyperlinks (Broder et al., 2000) and network size $100,000$ nodes. To obtain the confidence intervals for the results, for each algorithm, we consider 100 network samples to be generated and tested in the CSN statistical analysis framework. Figure 4.2 and 4.3 show the average in-degree and out-degree distributions of 100 network samples generated by each algorithm respectively. In these two figures, we plot the complementary cumulative distribution function for node degree distributions, since the CDF is better than than the PDF against the fluctuations on sample sizes on the visual form (Clauset et al., 2009). From Figure 4.2 the in-degree distribution of each algorithm looks roughly straight on a log-log plot. But in the tail of distribution, algorithms A, B and C do not have large in-degree nodes as those in the KRR and the KRRNoLM algorithms. In Figure 4.3 the out-degree node distributions of A, B and C algorithms bent downward when the out-degree value gets bigger than 100. From the plot of the node out-degree distribution, algorithms A, B and C seem not scale-free. We keep this in mind, and look at the results from the statistical test aspect, for each algorithm respectively.

4.3.1 Network size

First of all we want to know how the network size impacts on the scale free feature, because in theory it requires both the total number of nodes and the in-degree and out-degree to go to infinity for the power law distribution to be guaranteed (Bollobás et al., 2003; Krapivsky et al., 2001). Therefore we study how the scale free feature evolves with the network growth. We conduct the tests on dynamic networks with network growth from 100 to $10,000$ nodes. For each algorithm 100 random network samples are studied in the test. Figure 4.4 and 4.5 show the test results of in-degree and out-degree distributions for different algorithms. The confidence intervals plotted in Figures 4.4 onwards are at 95% confidence level. For consistency we always label the curves in the same order as the algorithms introduced before. According to Equation 4.1, if the test result

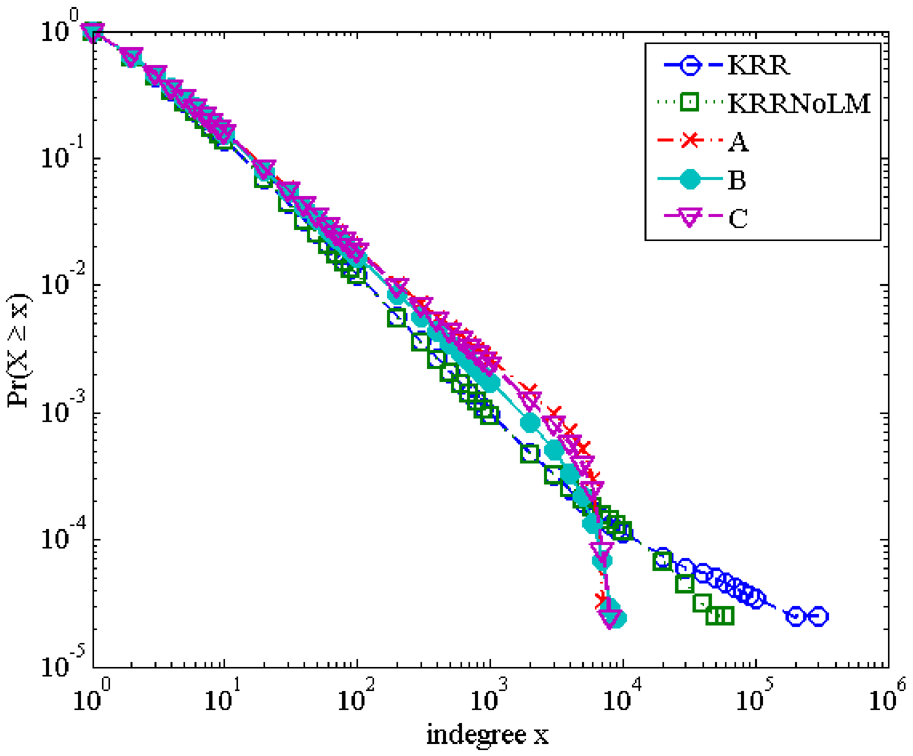

Figure 4.2: The average complementary cumulative in-degree node distribution of networks generated by various algorithms, versus the in-degree of the networks with $100,000$ nodes

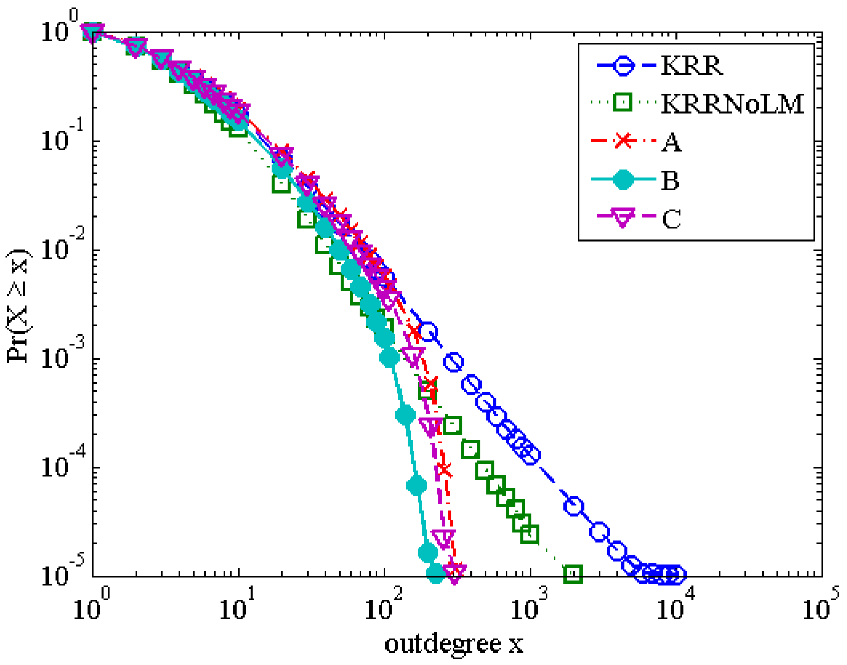

Figure 4.3: The average complementary cumulative out-degree node distribution of networks generated by various algorithms, versus the out-degree of the networks with $100,000$ nodes

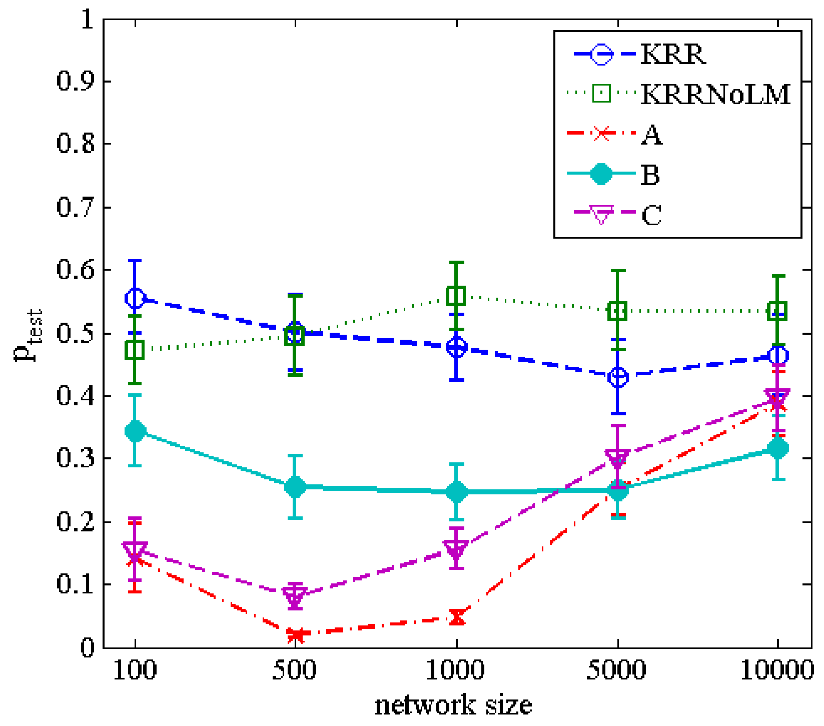

Figure 4.4: The statistical test results of in-degree node distribution of networks generated by various algorithms ($p = 0.1333, \lambda = 0.75, \mu = 3.55$)

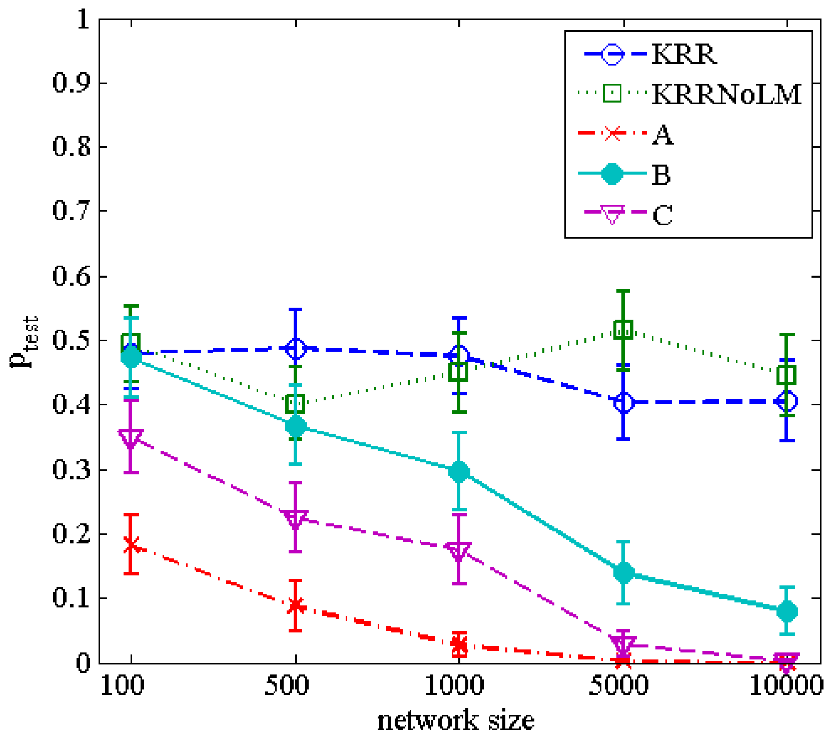

Figure 4.5: The statistical test results of out-degree node distribution of networks generated by various algorithms ($p = 0.1333, \lambda = 0.75, \mu = 3.55$)

p_{test} is greater than 0.1, a power law distribution hypothesis is plausible. Based on that criterion, KRR and KRRNoLM algorithms passed the test in both in-degree and out-degree distributions. The KRR algorithm has already been proven to be power law, so the KRR algorithm test results can be viewed as a reference and show that it already holds during growth, not just at infinity. When the network size gets bigger, the KRRNoLM algorithm results slightly change and close to 0.5 in both Figure 4.4 and 4.5. It seems to imply that in the KRRNoLM algorithm the network still maintains the power law distribution feature with multiple arcs and self loops removal. The key insight gained from the tests for the KRRNoLM algorithm is that removing the multiple arcs and self-loops after the entire network is generated does not affect the scale-free character. Also the scale-free feature exists on small networks, even with 100 nodes, generated by the KRR or the KRRNoLM algorithm.

Now let us analyse algorithms A, B and C. First look at the test results of the in-degree distribution shown in Figure 4.4, the algorithm B test results remain greater than 0.1 with the size of the network, and the test results of algorithms A and C are above 0.1 when network is large enough. It implies that algorithms A, B and C seems scale-free in the in-degree distribution. Then in Figure 4.5 showing the out-degree distribution, all three algorithms actually get worse with network size increase. In the preferential attachment rule, nodes with high out-degree are likely connected to nodes with high in-degree (Krapivsky et al., 2001). Algorithms A, B and C redraw an arc until it is neither a multiple arc nor a self loop. This behavior increases the amount of nodes with a few out-degree/in-degree and prevents nodes with extreme high out-degree to emerge, even in bigger networks. From Figure 4.5, we can see that algorithms A, B and C's test results p_{result} are below 0.1 even the network grows to $10,000$ nodes. It implies that none of algorithms A, B, C generates networks with the power law out-degree distribution.

4.3.2 Different probability values

Apart from testing the network size, we also consider the tests of each algorithm with different probability p values from 0.1 to 0.7. Here p can be viewed as the speed of adding a new node in the network. Within the same time period, the higher probability p, the more nodes are introduced in the network, and the few-

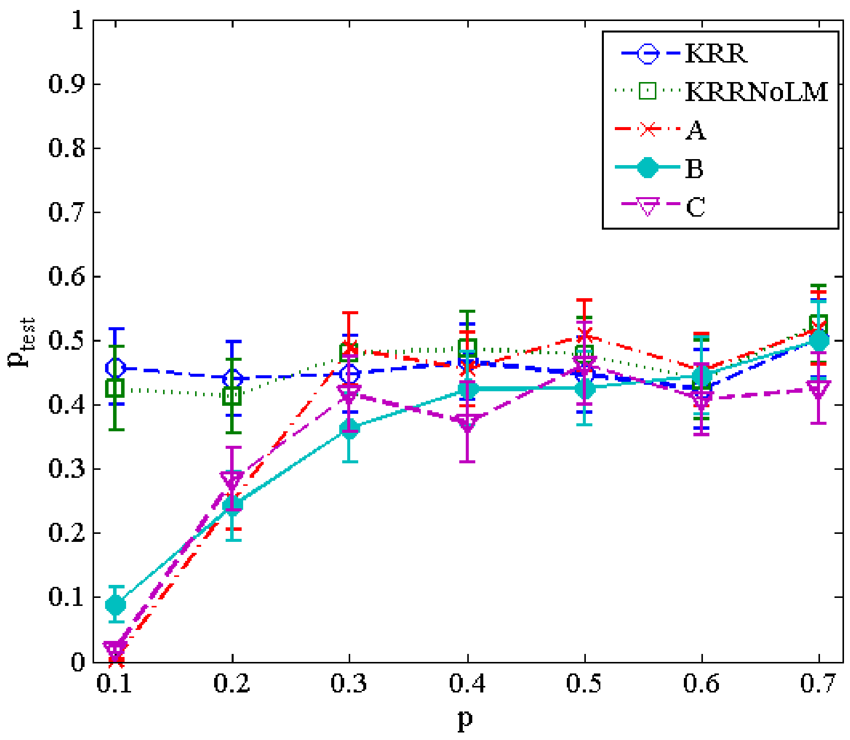

Figure 4.6: The statistical test results on in-degree distribution of networks generated by algorithms with various p ($\lambda = 1, \mu = 1$)

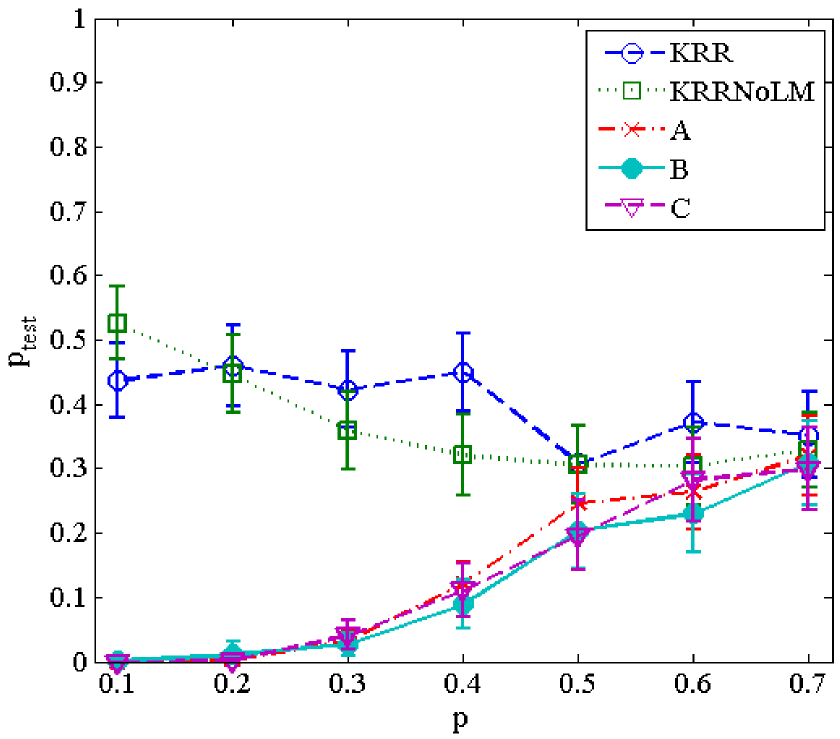

Figure 4.7: The statistical test results on out-degree distribution of networks generated by algorithms with various p ($\lambda = 1, \mu = 1$)

er arcs are added between existing nodes. Therefore, the chance of choosing multiple arcs and self loops is less. If p is high enough, we will expect all the algorithms will behave in a similar way. We set parameters $\lambda = 1$, $\mu = 1$ and network size $10,000$ nodes.

From Figure 4.6, we can see that when the value of p increases, p_{test} value of algorithm A increases and is greater than 0.1. As p increases to 0.2, test results of algorithms A, B and C are increased and above 0.1, and from $p = 0.2$ onwards, the test results of each algorithm remain above 0.1. These results indicate that algorithms A, B and C behave in a similar way if p is high enough. For algorithms KRR and KRRNoLM, as p increases, the test results remain reasonably large (> 0.1). We expect with high enough probability all the algorithms test results will converge to the same point. This is illustrated in Figures 4.6 and 4.7. It can be observed that A, B and C algorithms are plausible proved as a power law model when probability p value is large enough (0.7). The insight gained from the figures is that all algorithms are plausible to be scale-free if p is high enough.

4.3.3 Changing parameter values

We also studied the tests with parameters λ, μ changing. Parameters λ and μ are introduced in Equation (3.4), where the weights are defined to be used in adding arcs. We recap it. In the process of growing a network, introduce, at time t, there are n nodes for $i = 1, \dots, n$ the following weights: $v_i = \mu + O_i, w_i = \lambda + I_i$, where O_i is the out-degree of node i, I_i is the in-degree of node i, λ and μ are constants: $\lambda > 0$ and $\mu > -1$. If a new arc is added, it will be from node i to node j is given by $\frac{(\mu+O_i)(\lambda+I_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n}((\mu+O_i)(\lambda+I_j))}$. Here the constants λ and μ ensure nodes with small in-degree/out-degree have a chance to be attached or gain new arcs. For instance, take $\lambda = 0$, nodes with in-degree equal to 0 remain without incoming arcs.

Different λ: In this part, we only change the value of λ and set other parameters $\mu = 1$, $p = 0.1333$ and network size $10,000$ nodes. We vary the value of λ from 1 to 1000. With large value of λ, the choice of destination node doesn't depend on the in-degree value any more. That is because when the value of λ is much bigger than the in-degree value, the weight w_i in the above equation is mainly determined by λ rather than the node's in-degree value I_i. Since the value of

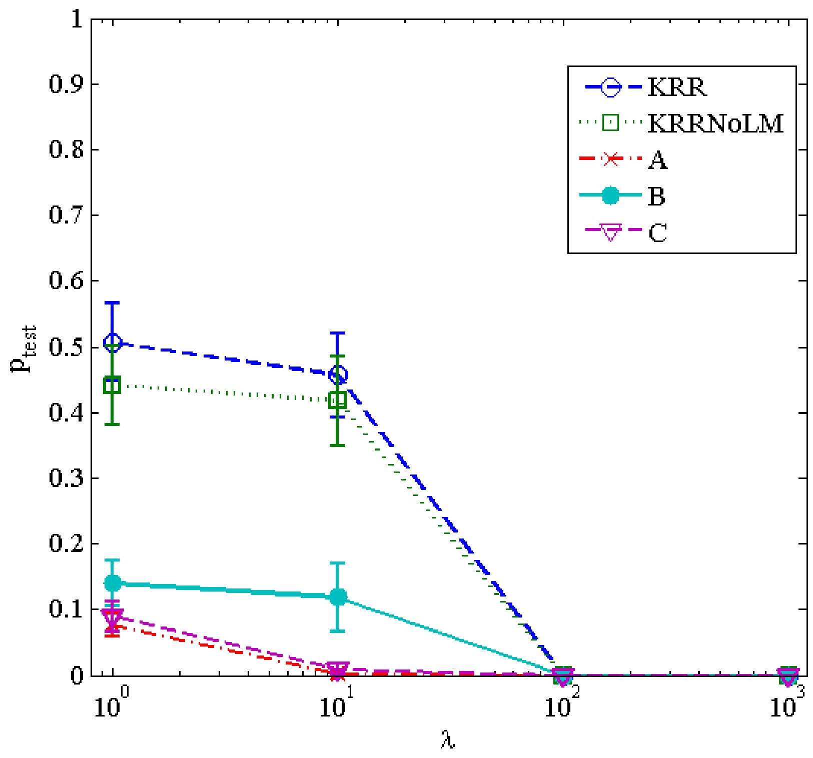

Figure 4.8: The statistical test results for in-degree distribution of networks generated by various algorithms with different λ

μ is fixed and the choice of origin node mainly depends on its own out-degree value, we will expect the out-degree might still hold a power-law distribution. While about in-degree distribution, we expect in-degree distribution is power law only when λ is small enough. Figure 4.8 presents test results in node in-degree distribution of networks generated by various algorithms. It can be observed that the test result p_{result} values of Algorithms KRR and KRRNoLM are greater than 0.1 until λ reaches 100. The test result of algorithm B is close to 0.1 until λ reaches 10. From $\lambda = 100$ onwards, the test results of algorithms KRR, KRRNoLM and B remain 0. For algorithms A and C, the test result p_{test} goes down and close to 0 till λ reaches 10. From $\lambda = 10$ onwards, the test results remain 0.

We note that for the KRR algorithm, the node in-degree distribution is power law with scaling parameter $\alpha = 2+p\lambda$ (see Equation (3.18)). When λ gets larger, the value of α is far beyond 3. Let p be fixed, if λ is larger than some certain value, the scale-free feature does not exist on networks generated using the KRR algorithm.

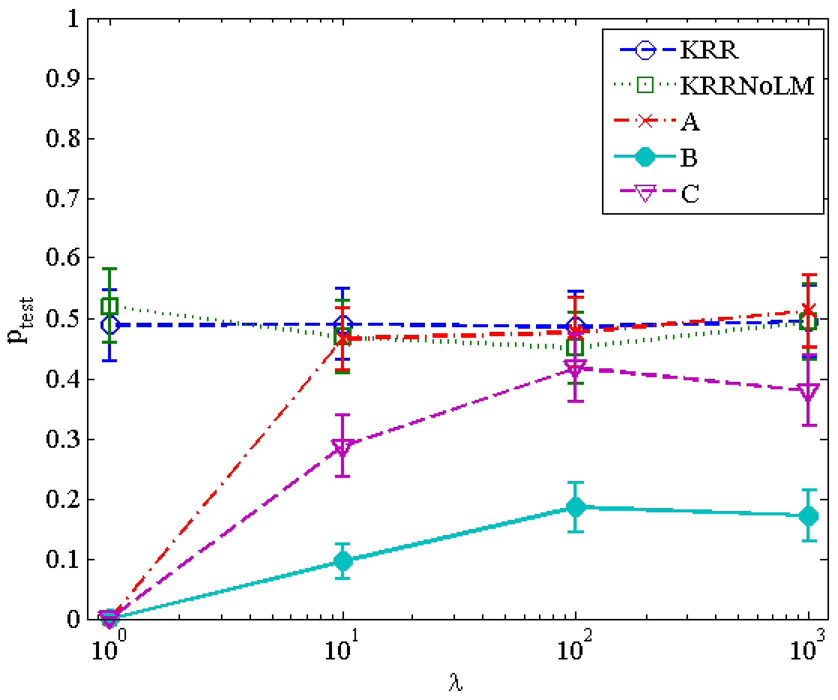

Figure 4.9: The statistical test results for out-degree distribution of networks generated by various algorithms with different λ

The test results for out-degree distribution are presented in Figure 4.9. It can be seen that the test results of the KRR and KRRNoLM algorithms are mostly unchanged and the values remain close to 0.5, compared to algorithms A, B and C. For the algorithms A, B and C, the test results increase and are above the threshold with the increase of λ. The explanation could be that when adding new arcs between two existing nodes, the choice of destination node are identically distributed in the overall network. Then choosing multiple arcs or self loops occurs much less in this situation. From $\lambda = 10$ onwards, algorithm A is similar to the KRR and KRRNoLM algorithms.

Different μ: We set $\lambda = 1$ and increase μ value from 1 to 1000. Other parameters are set the same values as those in the experiments in the different λ part. The test results for each algorithm are presented in Figures 4.10 and 4.11. This time, with μ increasing and becoming larger than the out-degree value, the out-degree value of a node is not influenced by its own out-degree value any more. When μ is much larger, the choice of origin node for a new arc between two existing nodes is identically distributed in all existing nodes rather than follow preferential attachment. We expect that out-degree distribution for all the algorithms do not follow a power law when μ becomes large enough. Now we l-

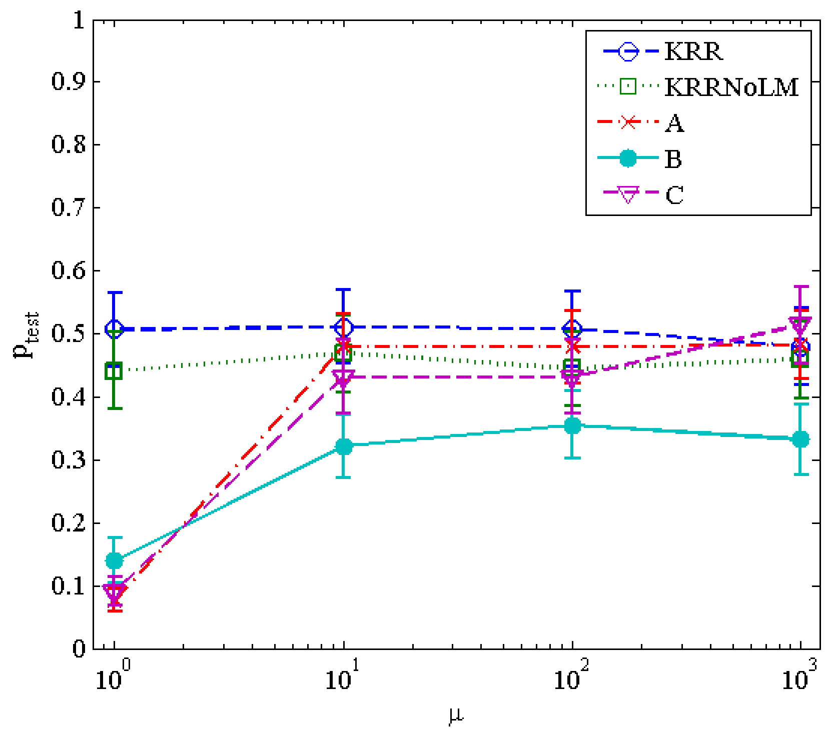

Figure 4.10: The statistical test results for in-degree distribution of networks generated by various algorithms with different μ

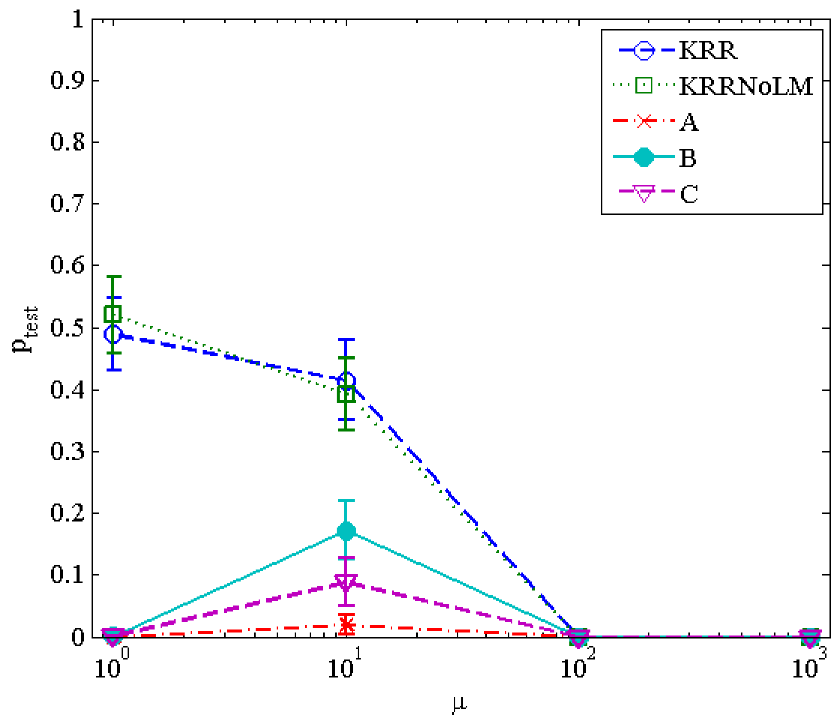

Figure 4.11: The statistical test results on out-degree distribution of networks generated by various algorithms with different μ

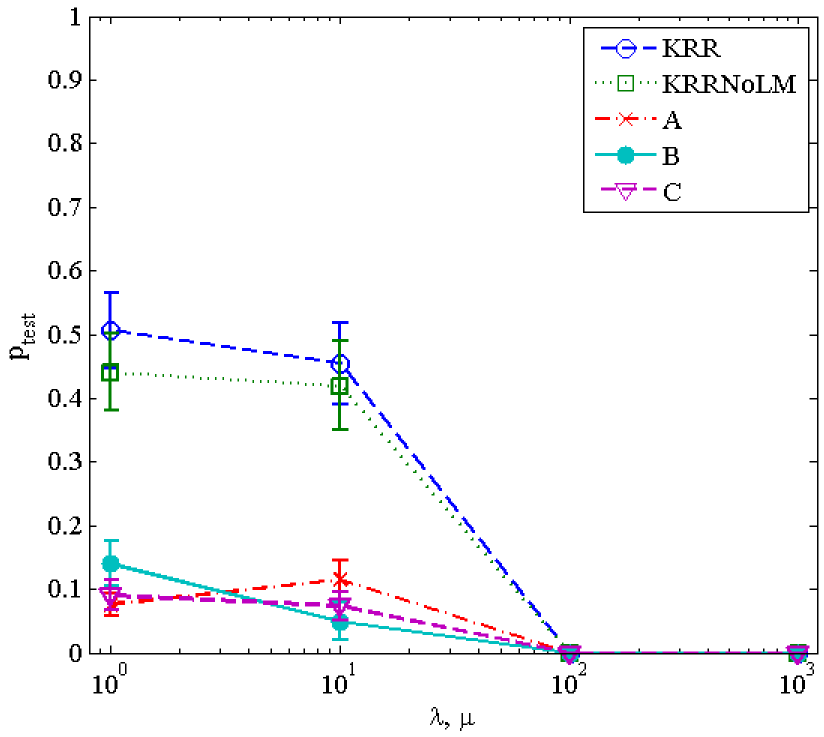

Figure 4.12: The statistical test results for in-degree distribution of networks generated by various algorithms with different λ and μ

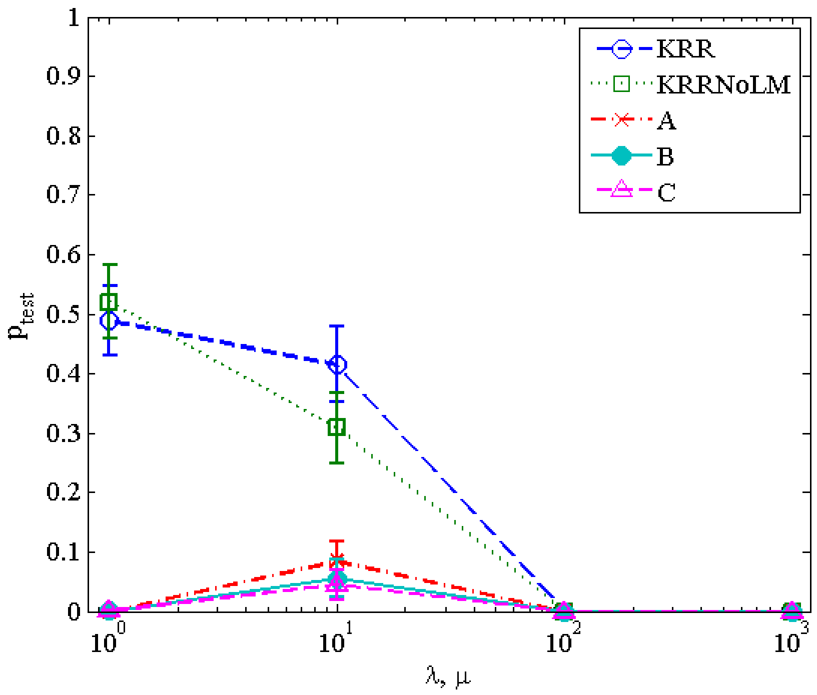

Figure 4.13: The statistical test results for out-degree distribution of networks generated by various algorithms with different λ and μ

ook at out-degree distribution test results in Figure 4.11. It can be seen that the test results of algorithms KRR and KRRNoLM are greater than 0.1 until μ reaches 100. From $\mu = 100$ onwards, the test results for each algorithm are close to 0.

Figure 4.10 presents the in-degree distribution test results for all different algorithms. One can see that the test results for algorithms KRR and KRRNoLM are similar and close to 0.5 for each μ value. It can also be seen that, for Algorithms A, B and C, the test results are greater than 0.1 when μ becomes large. That means the in-degree distribution for algorithms A, B and C is power law when μ is large enough (10 in this case).

Different λ and μ: In this test, we consider to vary both λ and μ. Figure 4.12 and 4.13 present the in-degree distribution and out-degree distribution test results for each algorithm, respectively. From figure 4.12, it can be seen that the test results of algorithms A, B and C are close to 0.1 until λ and μ reach 100, while the test results of algorithms KRR and KRRNoLM are close to 0.5. From $\lambda = 100$ and $\mu = 100$ onwards, the test results of all the algorithms are close to 0. The results on Figure 4.13 is similar to those in Figure 4.12. That is because with large value of λ and μ, the choice of both the origin node and destination node does not follow preferential attachment at all. Therefore, when λ and μ become large enough, none of all the algorithms are power law.

4.4 Empirical Data Results

4.4.1 Statistical results

Apart from studying experimental networks, we considered the empirical data as well. We realize OpenPGP Web of trust network as a good case study of trust networks exist in real world. The empirical data is collected from the Wotsap project (Cederlof, 2007), where keeps daily track of the largest strongly connected set of OpenPGP keys. In graph theory, a strongly connected component of a directed graph is a subgraph where there is at least one directed path between every node pair. We downloaded a snapshot of the key database of 28th June 2011 from Wotsap project as our data set. Figures 4.14 and 4.15 show the largest strongly connected component (LSCC) of OpenPGP network node in-degree and out-degree distributions with the estimated power law parameter values.

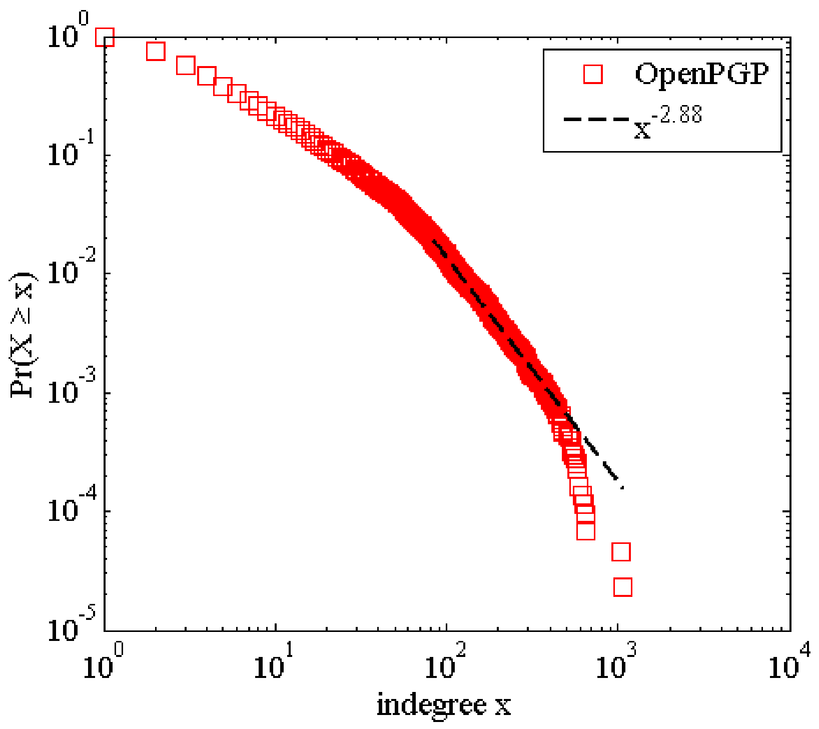

Figure 4.14: node in-degree distribution in LSCC of OpenPGP network on 28th June 2011

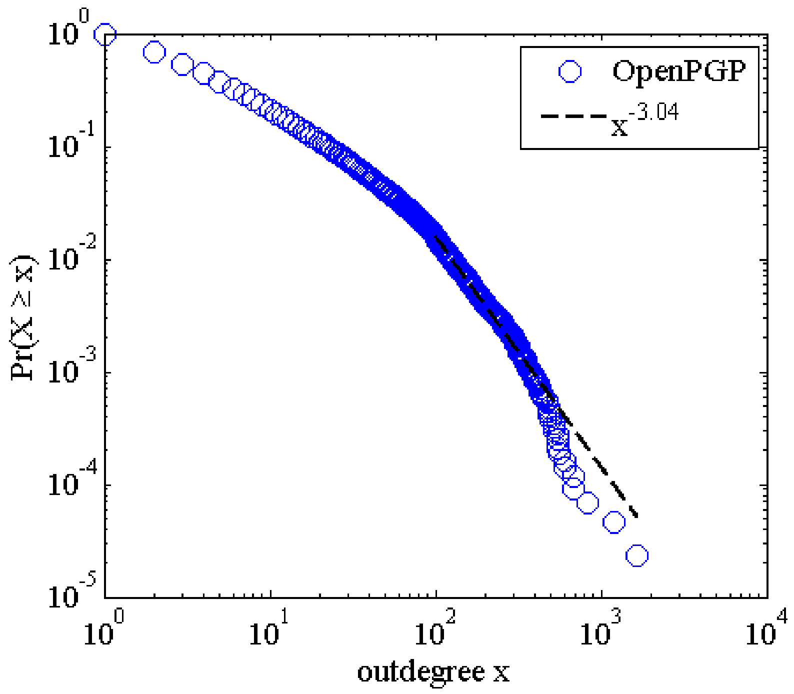

Figure 4.15: node in-degree distribution in LSCC of OpenPGP network on 28th June 2011

We conduct the statistical test on the empirical data in LSCC of OpenPGP Web of Trust network. The statistical results are listed in the table 4.1. According to the threshold of 0.1, a test result below this value will rule out the plausibility of a power law distribution. From the test results in the table, we can see that in the empirical data from the largest strongly connected set of OpenPGP web of trust network, the node out-degree distribution follows a power law but node in-degree does not have a power law distributed. In (Ulrich et al., 2011), the authors also employed the same testing techniques on the data set they collected from OpenPGP key database in December 2009 and found that none of node in-degree and out-degree in LSCC of OpenPGP web of trust network follow a power law distribution. New finding questions the web of trust network following a power law distribution. Although the OpenPGP web of trust network does not follow the power law form, it is the best approximation we know.

Degree	α	x_{min}	p_{test}
in-degree	2.88	78	0.001
out-degree	3.04	93	0.122

Table 4.1: Statistical results of the Empirical Data (28th June 2011)

4.5 Summary

In this chapter, to demonstrate the power law distribution of all the algorithms, an evaluation was carried out using statistical tests. From the test results of the experimental data, several key insights are gained. The KRRNoLM is the best under most circumstances. Removing the multiple arcs and self loops after an entire network has been generated does not affect the scale-free character, but the cost of the growth nature of the algorithm. We also found that the scale-free feature can exist on networks even with hundred nodes under some circumstances. All algorithms are plausible to be scale-free if p is high enough. From the test results of changing the values of parameters λ and μ, we found that none of algorithms hold the scale-free feature if parameters λ and μ are high enough. From the test results on the empirical data, we find that the web of trust network exhibits a power law distribution for the out-degree with the estimated parameter value 3.04. Although the OpenPGP web of trust network does not follow a power law in the strict sense, the power law is the best approximation we know.

Chapter 5

Fast Algorithmic Implementation

5.1 Introduction

In the previous chapter, several algorithms to generate directed scale-free networks have been studied. Being able to reproduce the properties of real networks allows much more accurate evaluation and simulation of routing algorithms and applications. To evaluate novel protocols through discrete-event simulation one first needs to generate networks of nodes and relationship between nodes. To obtain reliable results with small enough confidence intervals, one needs to generate many of these networks and it is therefore of importance that one is able to generate networks with the desired properties in reasonable time.

In this chapter, we will turn our attention to developing efficient implementations of the growth algorithms introduced in Chapter 3. The implementation of network generating algorithm poses implementation challenges. Firstly, every potential arc gets assigned a weight that may change when every arc is added. Secondly, once the weights are determined, selecting the next arc corresponds to weight random sampling, which in general requires computational effort similar to a linear search. We will therefore design and implement a set of algorithms and compare them with respect to computation time and memory consumption, in terms of both theoretical complexity analysis and experimental results. We will show through experiments that with the fastest algorithms networks with a million or more nodes can be generated in mere seconds.

In this chapter, we derive an algorithm that resolves both the problem of updating weights and of linear search. In what follows we first recap network growth algorithms in Section 5.2. In Section 5.3 we then introduce implementation algorithms. In Section 5.4 we analyse the complexity of the algorithms, theoretically as well as experimentally.

5.2 Weights

We recap the network algorithms introduced in Chapter 3. Each arc gets assigned to a weight. In particular, for any node i, let I_i be the in-degree of node i and O_i the out-degree of node i, then in KRR algorithm, weights $w_{i,j}$ are defined as follows:

$$w_{i,j} = (\mu + O_i)(\lambda + I_j), \tag{5.1}$$

where λ and μ are constants: $\lambda > 0$ and $\mu > -1$. The probability $p_{i,j}$ that, given an arc is added to the network, it is from i to j is then given by

$$p_{i,j} = \frac{w_{i,j}}{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j}}. \tag{5.2}$$

The above implies that arcs between nodes with a high number of incoming or outgoing arcs are more likely than between nodes with low number of arcs. Note that if an arc is added from a new node i to an existing node, that the out-degree O_i of the new node is zero, thus simplifying the weights to $w_{i,j} = \lambda + I_j$ (the constant μ can be omitted).

In algorithm A, which is a modified version of the KRR algorithm by not allowing multiple arcs or self loops, weights are defined as follows:

$$w_{i,j} = \begin{cases} (\mu + O_i)(\lambda + I_j) & \text{if} \quad i \not\to j \text{ and } \ i \neq j \\ 0 & \text{if} \quad i \to j \text{ or } \ i = j \end{cases} \tag{5.3}$$

The main difference between the implementation according to algorithm A and the KRR algorithm is that $w_{i,j}$ is set to 0 in algorithm A if an arc already exists or if it is a loop. We can see the fact that we disallow multiple arcs and self loops introduces several aspects that make it difficult to scale algorithm to large networks. We therefore design and implement a set of algorithms, using various ideas to reduce computation time as well as memory consumption compared to a 'Base' algorithm–we term these ideas 'Node Weights', 'Node Weights with Subtraction', 'Multi-sampling' and 'Reversed Look-up', respectively, and introduce these in the next section.

5.3 Generator Algorithms

Algorithm A poses considerably more challenging to implement in an efficient manner. We will see a main challenge in making algorithm scalable lie in the need to update weights $w_{i,j}$ in Equation (5.3). As a consequence, a straightforward implementation (termed the Base algorithm in Section 5.3.1) requires considerable weights update after an arc is added. This issue we address first. A second challenge is the selection of the arc once the weights are updated. This we can only resolve in one specific algorithm, namely Multi-sampling, as we discuss in Section 5.3.4 when we introduce Reversed Look-up.

For all algorithms we will compute the time and memory complexity. In the complexity analysis we ignore updating of the in-degree and out-degree of a node with every added arc–this has to be done in all algorithms. Similarly, we do not consider the storage in memory of the actual networks with all its nodes and arcs. This also has to be done in all cases, and it should be noted that storage of the N nodes and $\frac{N}{p}$ arcs is of dominant order in all but the Base algorithm.

5.3.1 Base implementation

The Base method is straightforward: store all elements $w_{i,j}$ in a matrix of size $N \times N$ and update these weights after every addition of an arc. In particular, if arc $x \to y$ is added, then $w_{x,j}$ needs to be updated for $j = 1, \ldots, n$, and $w_{i,y}$ needs to be updated for $i = 1, \ldots, n$. That is, a complete row and a complete column in the matrix needs to be updated. In addition, $w_{x,y}$ needs to be set to 0.

To make this precise, we introduce the superscript + to denote the updated weights when an arc is added. Similarly, we represent the increase of the out-degree of a node i as O_i^+ and the increase of the in-degree of node j as I_j^+. Assume that $x \to y$ is the last arc added, then $O_i^+ = O_i + 1$ if and only if $i = x$ and $O_i^+ = O_i$ otherwise. Similarly, $I_j^+ = I_j + 1$ if and only if $j = y$ and $I_j^+ = I_j$ otherwise. Hence, when $x \to y$ is the added arc, the weights in Equation (5.3) need to be updated as follows:

$$w_{i,j}^+ = \begin{cases} w_{i,j} + \lambda + I_j & \text{if} \quad i = x \quad \text{and} \quad i \not\to j \\ w_{i,j} + \mu + O_i & \text{if} \quad j = y \quad \text{and} \quad i \not\to j \\ 0 & \text{if} \quad i = x \quad \text{and} \quad j = y \\ w_{i,j} & \text{otherwise} \end{cases} \tag{5.4}$$

This process of updating weights has the following time complexity. If an arc is added at iteration n, there are up to $2n - 1$ (a row and a column) weights updated. Since at each iteration $\frac{1}{p}$ arcs are added, the total time complexity is $\frac{1}{p}\sum_{n=1}^{N}(2n-1) = O(\frac{N^2}{p})$ updates. A matrix is used to store the weights, thus requiring $O(N^2)$ storage.

The second aspect to be considered is the time it takes to draw a weighted random number according to the probabilities in Equation (5.2). The base algorithm for weighted random sampling is to draw a random number r between 0 and 1 and add up probabilities $p_{i,j}$ in Equation (5.2) in order until the sum exceeds r. ('In order' may for instance be implemented through a double `for` loop: `for (i=1) to n do { for (j=1) to n do{. . . }}`)

This way of weighted random number generation has complexity similar to a linear search through a list of size $n \times n$: it requires on average $\frac{1}{p}\frac{n^2}{2}$ operations at iteration n, and summing this leads to the results for pyramid numbers, which implies that the resulting time complexity is $O(\frac{N^3}{p})$ operations in total. There is no additional required memory for this way of drawing weighted random numbers.

We note that some algorithmic tricks can be thought off to speed up the drawing of weighted random numbers, such as traversing the matrix backward when the random number is larger than 0.5 (for instance). This, however, does not change the order of the algorithm. Similarly, as we already remarked, if a new node is added, the out-degree of the new node is 0, and hence the weights in Equation (5.3) simplify. We exploit this in our implementations to make the algorithms more efficient when adding a node, but the order of the algorithm does not change because of it. We therefore will not discuss such issues in more detail.

5.3.2 Node weights

Since the Base algorithm stores the complete matrix of weights its memory requirement of $O(N^2)$ makes it less attractive for a large network size. Roughly speaking, modern day personal computers may be expected to hold up to 10^9 doubles in memory, thus limiting N to about 30,000.

The Node Weights method resolves this issue, by storing and updating weights per node, instead of per arc. This immediately implies that storage requirements will go down to $O(N)$. In particular, at iteration n, for each node i we store

$$w_i = \sum_{j=1}^{n} w_{i,j}. \tag{5.5}$$

The probability p_i that, given an arc is added to the network, it has i as the origin is then given by:

$$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i}. \tag{5.6}$$

Furthermore, once the origin is determined, the destination is determined by computing $w_{i,j}$ on the fly for given origin node i.

Important is that the node weights w_i can be updated without knowing the individual values $w_{i,j}$, because otherwise there would be no gain from maintaining node weights. This works in a similar manner as for the Base algorithm: if arc $x \to y$ is added we have again that $O_x^+ = O_x + 1$ and $I_y^+ = I_y + 1$, and we derive that the updated node weights w_i^+ obey:

$$w_i^+ = \begin{cases} w_x + \sum_{j|x \not\to j}(\lambda + I_j) - w_{x,y} & \text{if} \quad i = x \\ w_i + \mu + O_i & \text{if} \quad i \not\to y \text{ and } i \neq x, y \\ w_i & \text{otherwise} \end{cases} \tag{5.7}$$

Updating the weights by above equations takes order n operations for each arc in the n-th iteration, thus giving $O(\frac{N^2}{p})$ overall complexity for updating, as in the base case. However, selection of an arc through weighted random sampling is an order less expensive than in the base case. A sequential search is used to find the origin node, and for this origin node all possible destination nodes are considered. (Again, we sum up probabilities p_i and then probabilities $p_{i,j}$ until they sum to r. To make this more precise would lead to cumbersome explanation not necessary for the thrust of this paper.) As we remarked, for the chosen node i, we generate the weights $w_{i,j}$ on the fly from Equation (5.3) since we do not store the individual weights. The time complexity for the weighted random sampling in the Node Weights algorithm is thus $O(\frac{N^2}{p})$.

5.3.3 Node weights with subtraction

Node Weights with Subtraction is a variation of Node Weights in which we decrease the number of updates. From Equation (5.7) one sees that weights w_i are updated for every node i that is not connected to y ($i \not\to y$). In Node Weights with Subtraction we do the opposite, and update w_i if and only if $i \to y$. The main observation behind the method is that the node weights in Equation (5.5) can be rewritten as:

$$\begin{aligned} w_i = \sum_{j=1}^{n} w_{i,j} &= \sum_{j=1|i\not\to j, i\neq j}^{n} (\mu + O_i)(\lambda + I_j) \\ = \sum_{j=1}^{n} (\mu + O_i)(\lambda + I_j) &- \sum_{j=1|i\to j \vee i=j}^{n} (\mu + O_i)(\lambda + I_j) \\ = (\mu + O_i)(n\lambda + A_n) &- \sum_{j=1|i\to j \vee i=j}^{n} (\mu + O_i)(\lambda + I_j), \end{aligned}$$

where A_n is the total number of arcs at iteration n. For each node, we then keep track of the term $(\mu + O_i)(n\lambda + A_n)$ (which we can easily track and update) as well as of $\sum_{j=1|i\to j\vee i=j}^{n}(\mu + O_i)(\lambda + I_j)$. Since the latter term has fewer elements in the sum it is less effort to update that term than it is to update the actual values w_i (as in the Node Weights method).

We will not write down the equivalent of Equation (5.7) to update the elements $\sum_{j=1|i\to j\vee i=j}^{n}(\mu + O_i)(\lambda + I_j)$. Note that although it can be expected that the

Node Weights with Subtraction method is more efficient than Node Weights, the time and memory complexity orders do not change.

5.3.4 Multi-sampling

The idea behind Multi-sampling is radically different from the previous approaches in that updates are no longer carried out. Instead of enforcing that no multiple arcs or loops will be generated by setting weights to 0, Multi-sampling allows an arc to be selected that leads to multiple arcs or a loop, but then ignores it and retries the sampling for an arc. We will see that this implies a constant computational complexity for updates in each iteration, but that it increases the number of samples, thus resulting in $O(\frac{N}{fp})$ computational complexity, where $\frac{1}{f}$ is the average number of samples per iteration.

The algorithm works as follows. Introduce, for $i = 1, \ldots, n$ the following weights:

$$\begin{aligned} v_i &= \mu + O_i, \\ w_j &= \lambda + I_j. \end{aligned} \tag{5.8}$$

Then generate an arc $x \to y$ by conducting weighted random sampling using the weights v_i to determine x and by conducting weighted random sampling using the weights w_i to determine y. If $x \to y$ already exists or if $x = y$, then repeat the procedure until a new arc is added.

We now have to show that this procedure correctly implements our model, i.e., that it generates probabilistically equivalent networks as when the probability of adding an arc $x \to y$ is given by Equation (5.2). This follows directly from considering, for the Multi-sampling method, the conditional probability $p^{\mathrm{MS}}_{x,y}$:

$$\begin{aligned} p^{\mathrm{MS}}_{x,y} &= Prob\{\text{arc } x \to y \text{ is added } | \text{an arc is added}\} \\ &= \frac{Prob\{\text{arc } x \to y \text{ is added } \wedge \text{ an arc is added}\}}{Prob\{\text{an arc is added}\}} \end{aligned}$$

If $x \to y$ already exist, then an arc is not added, and so the numerator evaluates to false, resulting in $p^{\mathrm{MS}}_{x,y} = 0$ if $x \to y$ already exists. Similar, if $x = y$, an arc is not added and $p^{\mathrm{MS}}_{x,y} = 0$. If $x \to y$ does not yet exist, then an arc is added and the numerator becomes

$$Prob\{\text{arc } x \to y \text{ is added}\} = \frac{v_i w_j}{\sum_{i=1}^{n} \sum_{j=1}^{n} v_i w_j}.$$

The denominator equals:

$$Prob\{\text{an arc is added}\} = \frac{\sum_{i=1}^{n} \sum_{j=1 | i \not\to j, i \neq j}^{n} v_i w_j}{\sum_{i=1}^{n} \sum_{j=1}^{n} v_i w_j}.$$

As a result we obtain for the conditional probability that given an arc is added, it is arc $x \to y$:

$$p_{x,y}^{\text{MS}} = \frac{v_i w_j}{\sum_{i=1}^{n} \sum_{j=1 | i \not\to j, i \neq j}^{n} v_i w_j}. \tag{5.9}$$

Hence, filling in $w_{i,j}$ in Equation (5.2) and v_i and w_j in Equation (5.9) we find that probabilistically Multi-sampling generates networks consistent with our model.

For the performance of the Multi-sampling method it will be important to determine the amount of resampling that is required. Every time an arc $x \to y$ is selected that already exists or $x = y$, a new attempt must be made (which involves drawing a new random number, and selecting an arc according to the procedure under Equation (5.8)). To determine the average number of resamples, let f be the probability the sample is successful:

$$f = \frac{\sum_{i=1}^{n} \sum_{j=1 | i \not\to j, i \neq j}^{n} v_i w_j}{\sum_{i=1}^{n} \sum_{j=1}^{n} v_i w_j}. \tag{5.10}$$

Then the average number of tries until a sample is successful equals $\sum_{k=1}^{\infty} k(1 - f)^{k-1} f = \frac{1}{f}$. Obviously, the required number of samples gets high if the success probability f is small. Since f is a potential bottleneck for the Multi-sampling methods Section 5.4.3 shows experimental results for f.

Finally, we note that the storage requirements for Multi-sampling only involve maintaining weights v_i and w_j in Equation (5.8). These weights can easily be computed from the already stored in- and out-degrees, and therefore there is no specific storage needed for the weights.

5.3.5 Reversed look-up

Thus far the algorithms have dealt with the issue of updating weights, either decreasing the storage needed for the weights or the time required for updates. However, considerable computational effort is also required to sample weighted random numbers once the weights are established. For the Multi-sampling approach, however, the weights are such that one can store the weights in such a way that, given a random number, the appropriate weighted random number can directly be read from the data structure.

We call this idea 'Reversed Look-up', and it has its origin in a commonly proposed algorithm for weighted random sampling if all weights have integer values (see for instance (jimt, n.d.) and also the BA algorithm implementation in Peersim (Jesi, 2004) for examples of this and related ideas). In our case, following Equation (5.8), in iteration n weighted random sampling selects node i with probability p_i defined as:

$$p_i = \frac{v_i}{\sum_{i=1}^{n} v_i} = \frac{\mu + O_i}{\sum_{i=1}^{n}(\mu + O_i)} = \frac{\mu + O_i}{n\mu + A_n}, \tag{5.11}$$

where A_n is the total number of arcs at iteration n. We will now first deal with the constants μ, before applying Reversed Look-up to the integer-valued out-degrees O_i.

Let r be a random number between 0 and 1. To deal with the constant μ, we select node $x = 1, \ldots, n$, if $\frac{\mu(x-1)}{n\mu+A_n} \leq r < \frac{\mu x}{n\mu+A_n}$. This means that if $r < \frac{n\mu}{n\mu+A_n}$ the origin node for the new arc is selected uniformly from all n nodes, since μ contributes the same constant value to any probability p_i. If, on the other hand, $r \geq \frac{n\mu}{n\mu+A_n}$, the outgoing arc is not yet decided and the Reversed Look-up comes into effect, as follows.

We maintain an array of size A_n with integer values, such that in each array element a node number is stored. More precisely, the array has O_i elements with value i. We then select any of the array elements with equal probability–since there are O_i array elements for node i the likelihood that a node is chosen is proportional to O_i (we make the correctness argument precise below).

To select an array element according to a uniform distribution, we first scale up the random number $r \geq \frac{n\mu}{n\mu+A_n}$ so it is a uniformly distributed number between 0 and A_n: $r_{\text{new}} = r * (n\mu + A_n) - n\mu$. Then we select array index k as $k = \lfloor r_{\text{new}} \rfloor$

(the floor operator $\lfloor\rfloor$ indicating rounding to the nearest lower integer). The origin node is then the value of the array element at index k.

Once the origin node is determined, the destination node is determined similarly by maintaining an array of $A_N = \frac{N}{p}$ elements with nodes based on in-degrees I_j.

To demonstrate the correctness of the approach, node x is selected according to a uniform distribution (that is, with probability $\frac{1}{n}$) if $r < \frac{n\mu}{n\mu+A_n}$ and with probability $\frac{O_x}{A_n}$ if $r \geq \frac{n\mu}{n\mu+A_n}$. Together, this means that node x is the origin with probability $\frac{1}{n}\frac{n\mu}{n\mu+A_n} + \frac{O_x}{A_n}(1 - \frac{n\mu}{n\mu+A_n}) = \frac{\mu+O_x}{n\mu+A_n}$, which confirms to Equation (5.11).

This idea of Reversed Look-up works because of the integer value of the weights and because updates can be implemented very simply. For instance, it is not easy (if at all possible) to efficiently implement Reversed Look-up when weights may decrease, such as in the Node Weights method. When using Multi-sampling updating the array is straightforward. Assume arc $x \to y$ is last added, then we add to the array for in-degrees one element with value y and to the array with out-degrees one element with value x.

The time complexity of the Reversed Look-up method is minimal. To update the array and read the right element from the array there are some operations each time an arc is added, resulting in time complexity $O(\frac{N}{p})$. Importantly, memory use is increased through the use of two arrays of $O(\frac{N}{p})$ elements.

We will see in the results section that the ability to use Reversed Look-up in the Multi-sampling approach dramatically reduces the computational effort to generate scale-free directed networks. We will see that networks with a million or more nodes are feasible.

5.4 Evaluation of Generator Algorithms

Before discussing our experimental results, let us recap the theoretical complexity results we derived in Section 5.3, by comparing the order of all algorithms in Table 5.1, for CPU time and memory consumption, respectively. In Table 5.1 the time complexity is divided in two: the time an algorithm takes in updating weights, and the time an algorithm takes in determining the correct arc to add based on weighted random sampling. Note that the CPU time complexity does not include

	CPU Time		Memory Use		
Algorithm	Updating weights	Selecting the arc	Weights	Selecting the arc	Network
Base	$\frac{N^2}{p}$	$\frac{N^3}{p}$	N^2	0	$\frac{N}{p}$
Node Weights	$\frac{N^2}{p}$	$\frac{N^2}{p}$	N	0	$\frac{N}{p}$
Node Weights with Subtraction	$\frac{N^2}{p}$	$\frac{N^2}{p}$	N	0	$\frac{N}{p}$
Multi-sampling	0	$\frac{N}{fp}$	0	0	$\frac{N}{p}$
Multi-sampling with Reversed Look-up	0	$\frac{N}{fp}$	0	$\frac{N}{p}$	$\frac{N}{p}$

Table 5.1: Orders of CPU and memory use for various algorithms

aspects that are required in all algorithms, such as updates of in-degrees and out-degrees. The memory use is given for the same two aspects: storing weights and memory used for selecting the arc through weighted random sampling. The table also shows the memory requirement for the generated network itself, since this is a dominant factor in the memory use of all algorithms.

We see from Table 5.1 in the first two columns on CPU time that we may expect that Multi-sampling will outperform the Base and Node Weight methods, unless the probability f becomes too small (f is the probability resampling is not needed, and thus $\frac{1}{f}$ is the expected number of samples for each added arc). We will show in our experiments that Multi-sampling is indeed the preferred method, and we will also experimentally show that the number of retries $\frac{1}{f}$ decreases with the number of iterations to a small, almost constant, value. The latter is important, since otherwise the method would break down because of excessive resampling.

We also see from the table that a potential bottleneck exists for Multi-sampling with Reversed Look-up in the use of memory to select the arc using weighted random sampling. After all, the point of the Reversed Look-up method was to trade memory for CPU. However, we will see that this use of memory is of the same order as that for storing the generated network itself, something all algorithms need to do. This implies that the base and Node Weight methods never really are competitive compared to Multi-sampling with Reversed Look-up: even if memory use increases in Multi-sampling with Reversed Look-up, the time the other algorithms take leaves them unattractive.

In what follows we analyze the results of our experiments. All the methods are implemented and executed within the Java Peersim simulation environment for peer-to-peer networks (Jesi, 2004). As a general-purpose P2P simulation environment Peersim is concerned with more than efficiency alone and one may expect some performance or memory usage overhead compared to bespoke implementations. Nevertheless, we are convinced that our implementation and experiments are indicative for typical use of the algorithms we developed.

To achieve fair results, all the experiments were conducted on the same machine. It has a Pentium(R) processor, with CPU 3 GHz and 2 GB of RAM. Effectively, we were able to use up to about 1.4 GB of memory in our experiments. All experiments were repeated up to fifty times to create tight confidence intervals. In general, results did not show much variance. Even in cases that computation time for each data point was too high (up to one day) to do many experiments, we still achieved relatively stable results. In order to collect the performance results, we used the Java method `System.currentTimeMillis()` to give current system time in milliseconds and added time stamps to the Java programming code to measure the time the algorithm needs to generate the networks. To measure the memory usage, we use two common Java methods `Runtime.totalMemory()` and `Runtime.freeMemory()` to calculate how much space has been occupied by the algorithm. Because memory allocation is managed by the Java Virtual Machine, the results may be influenced by the working of the JVM. However, we will see that the results can be satisfactory explained from our understanding of the working of the algorithms and implementation.

5.4.1 Baseline performance comparison

We first compare the performance of the various algorithms for typical settings. Since discrete-event simulation studies of peer-to-peer algorithms often concern networks of some ten thousands of nodes (e.g., (Lv et al., 2002; Zhang and van

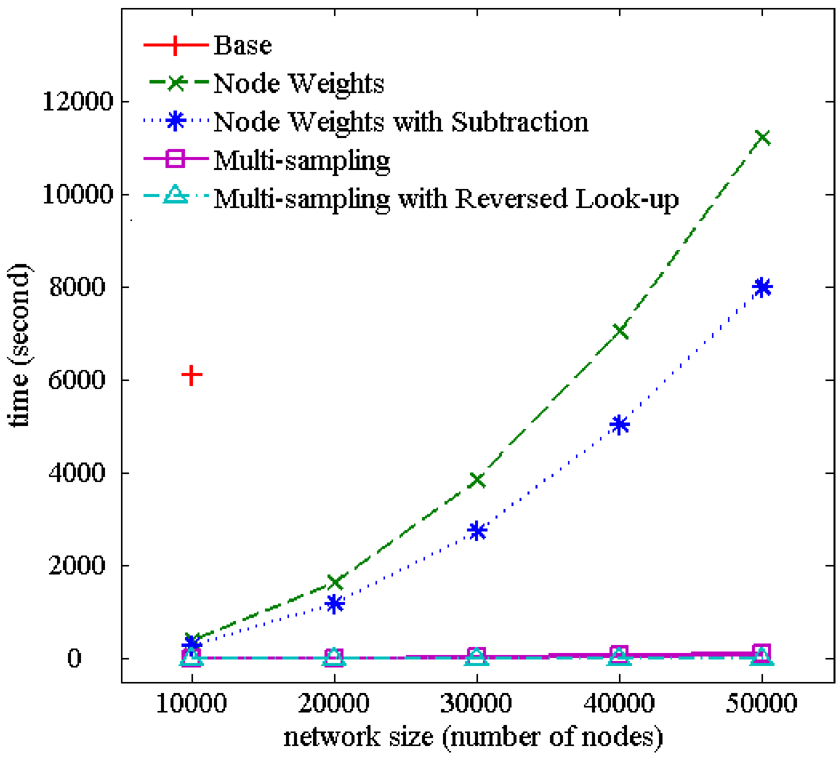

Figure 5.1: CPU time for average networks ($p = 0.1333$)

Moorsel, 2008)), we vary the network size N from $10,000$ to $50,000$ nodes. In addition, we use the parameter values derived in (Krapivsky et al., 2001) for the world-wide web (in turn attributed to data from (Broder et al., 2000)): $p = 0.1333$, $\lambda = 0.75$ and $\mu = 3.55$. Note that the values of λ and μ do only influence the CPU or memory use through the probability f in Equation (5.10), but that p is a very important factor for all metrics and methods.

Figure 5.1 shows CPU time for the various methods. To simplify the understanding of the figures, we note that we always label the curves in the order they appear in the graph (from top to bottom). In this case, the Base method only generated a single point for $N = 10,000$. The Base method does not complete for larger values of N because it runs out of memory. The Base method thus clearly performs worst. The two increasing curves in Figure 5.1 are for Node Weights without and with Subtraction, respectively. In fact, the increase is roughly quadratic in the number of nodes, as we expect from the complexity results in Table 5.1. Finally, the two Multi-sampling approaches easily outperform the others, both demonstrating an almost flat line.

The memory consumption for the different methods is displayed in Figure 5.2

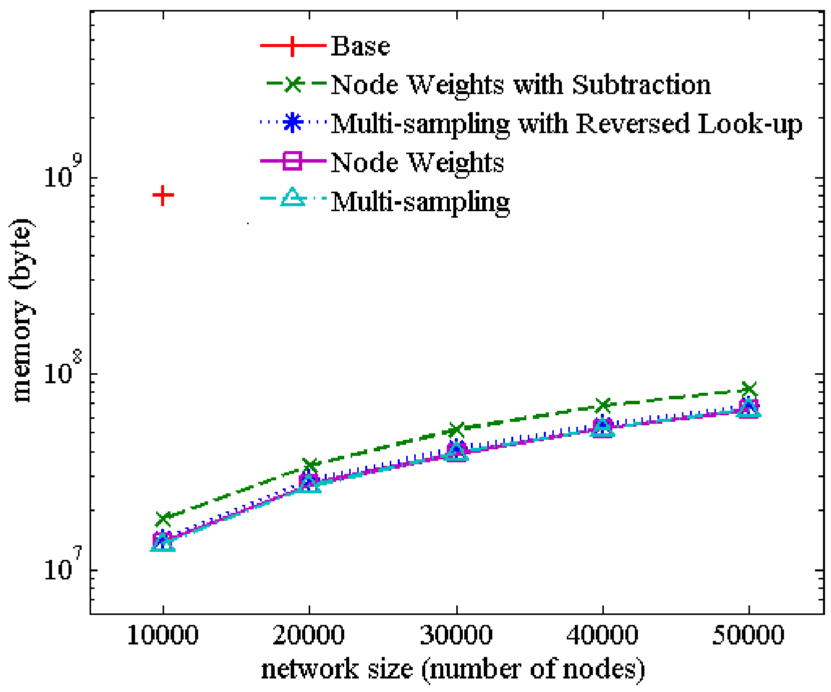

Figure 5.2: Memory use for average networks($p = 0.1333$, logarithmic scale)

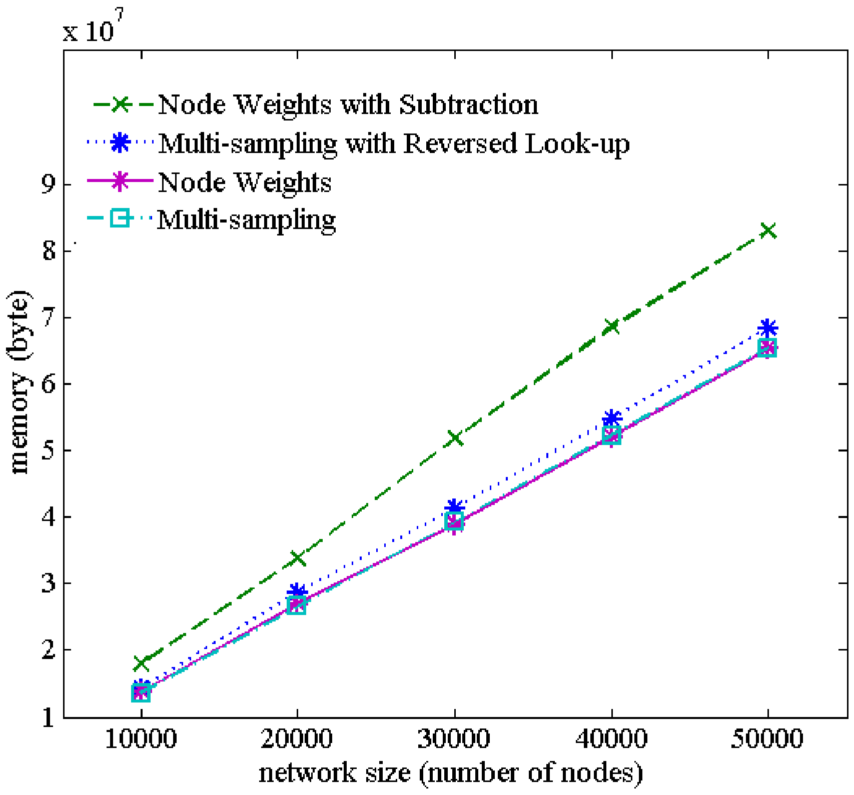

Figure 5.3: Memory use for average networks($p = 0.1333$, linear scale)

(notice the logarithmic scale) and Figure 5.3 (in linear scale). One sees that the Base method indeed runs out of memory. As we remarked, we can effectively use up to 1.4GB of memory, and for $N = 10,000$ the Base method uses close to 1.0GB already. The other approaches all exhibit similar memory consumption– this can explained from the fact that memory use in all cases is of order $\frac{N}{p}$, dominated by the storage of the network itself. Note that Multi-sampling with Reversed Look-up consumes slightly more memory than either Multi-sampling or Node Weights, but in essence all four methods are comparable.

In conclusion, we see that for a common range of parameter values, Multi-sampling with or without Reversed Look-up clearly outperform the other methods with respect to the required computation time. Although the Node Weights method is competitive when considering memory use, it does not dramatically improve over either Multi-sampling method. Therefore, the results suggest that the choice is be between the two variations of Multi-sampling. When generating networks with a million or more nodes in Section 5.4.4, we will discuss in more detail the CPU and memory implications of using Reversed Look-up or not (see Figures 5.9 and 5.10).

5.4.2 Highly connected networks (p small)

Since the complexity numbers in Table 5.1 all tend to infinity when the probability $p \downarrow 0$ we now research the performance of the various approaches when p gets small. For $N = 10,000$ Figures 5.4 and 5.5 show the CPU and memory results, respectively, for small values of p (note the logarithmic scales of the figures). For a network with N nodes, the average number of arcs is $\frac{N}{p}$ and the maximum number of arcs is $N(N-1)$. We thus see that realistically one should have p considerable larger than $\frac{1}{N-1}$. In our network of size $N = 10,000$ we let p get as small as 0.001.

Figure 5.4 shows that only Multi-sampling (with or without Reversed Look-up) can generate networks with $p = 0.001$. For instance, the Node Weight methods require more than 10 hours to generate networks for $p = 0.01$, and networks for smaller p can therefore not be generated in practice.

Even though Figure 5.5 demonstrates that the Node Weights method uses as little memory as Multi-sampling, we see that the conclusions from Section 5.4.1 do not

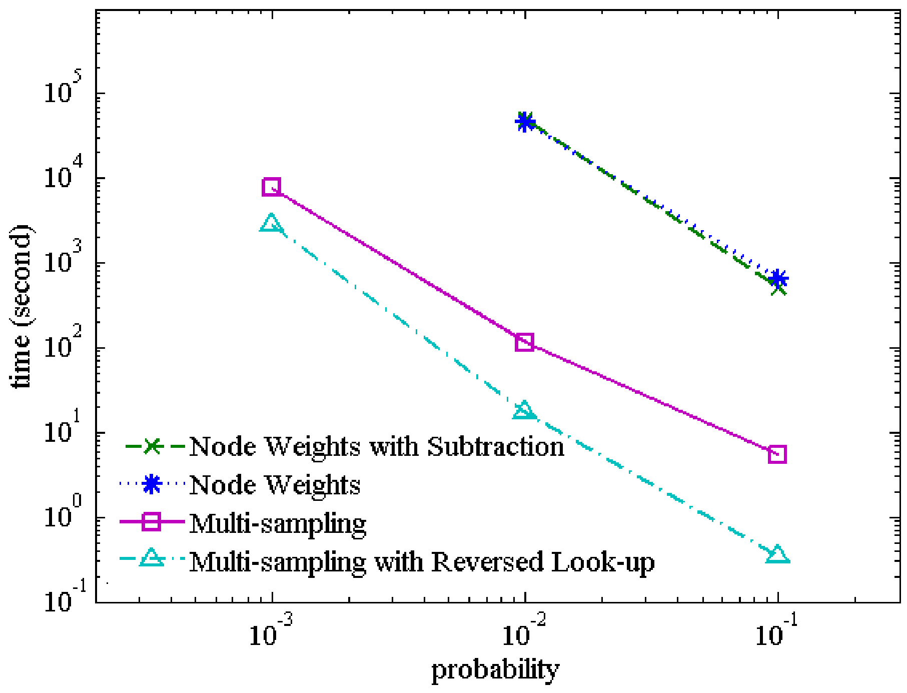

Figure 5.4: CPU time for highly connected networks ($N = 10,000$)

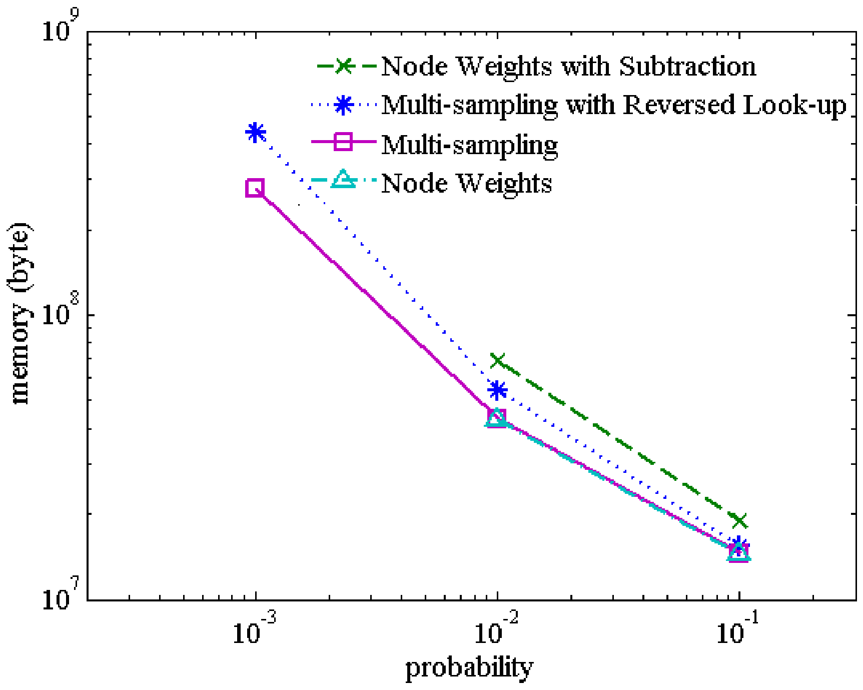

Figure 5.5: Memory use for highly connected networks ($N = 10,000$)

change significantly compared to the results for small p values in this section. Multi-sampling is so fast that the Node Weight methods cannot compete, even though Node Weight is memory efficient. Note that based on Table 5.1 this was not a foregone conclusion because of the unknown implications of the probability f (which relates to the required amount of resampling) in the Multi-sampling methods. We study this further in the next section.

5.4.3 Amount of resampling in multi-sampling methods

Table 5.1 shows the dependence of the CPU time needed for Multi-sampling methods on the probability f, which is the probability an arc is successfully added, as given in Equation (5.10). Unfortunately, we do not have expressions or convergence results for f. We therefore experimentally investigate f as a function of the iteration number.

We generate networks of up to $180,000$ nodes, count the number of resamples over intervals of $20,000$ nodes and divide it by the number of arcs added in these intervals. In doing so, we obtain $\frac{1}{f}-1$, the number of 'wasted' resamples for each successfully added arc. (We note that using the Node Weight methods we can numerically compute f precisely for any network, but the Node Weight method is prohibitively slow for the small values of p and large N considered. Hence, we used Multi-sampling with Reversed Look-up and sampled $\frac{1}{f}$, repeating the experiment sufficiently often to gain tight enough confidence intervals.)

Figure 5.6 presents the average number of resamples as a function of the iteration count. We show a curve for $p = 0.01$, $p = 0.1$ and $p = 0.1333$ (as in Section 5.4.1). Clearly, f depends very much on p, but in all cases the number of wasted samples is small. For instance, for $p = 0.01$ the number of resamples is less than 1 per succesfully added arc, while for more regular values such as $p = 0.1333$ the number of resamples is even less.

Importantly, however, all curves decrease as a function of the iteration count. This implies that for larger network the danger decreases that f becomes a bottleneck. This probably can be explained from the fact that with increasing iteration count n the number of existing arcs $\frac{n}{p}$ becomes less and less significant compared to the number $n(n-1) - \frac{n}{p}$ of not yet existing arcs. However, this depends on the actual values of the weights corresponding to existing arcs, so we

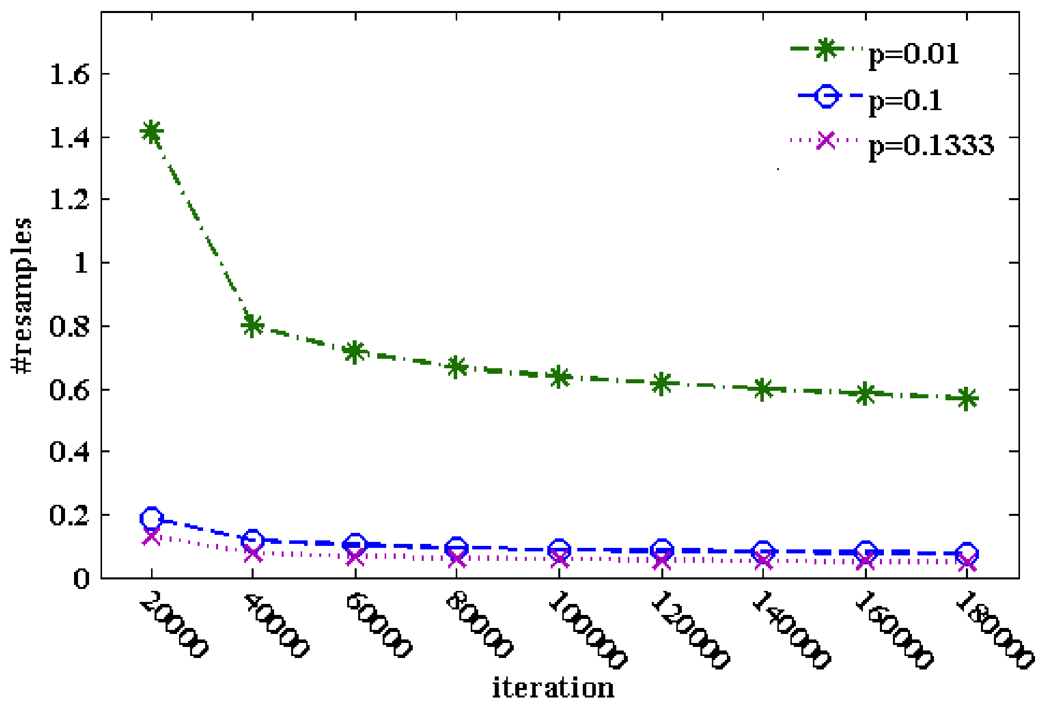

Figure 5.6: Number of resamples

only suggest it as a possible explanation that remains to be proven. Clearly, it would be of great interest to derive theoretical results for f or its convergence, but this is beyond the scope of our current presentation.

5.4.4 Networks with a million nodes

Finally, we want to push the algorithms and generate as large a network as we can using our current implementation in Peersim. We have already seen that only the Multi-sampling methods should be considered for average sized networks, and have identified that Multi-sampling with Reversed Look-up is superior in time, while plain Multi-sampling is more memory efficient. Figure 5.7 and Figure 5.8 confirm this for $N = 100,000$, and different values of p. More precisely, Reversed Look-up takes about 20% more memory for the whole range of p values (200MB for $p = 0.01$). On the other hand, computation time for Multi-sampling gets considerably worse, and can differ more than a factor 10. So, an early conclusion would be to exclusively use Multi-sampling with Reversed Look-up.

We see in Figures 5.9 and 5.10 how far we can push the two Multi-sampling methods. As far as CPU use is concerned Reversed Look-up is clearly beneficial, since Multi-sampling without Reversed Look-up takes close to 10 hours (Figure

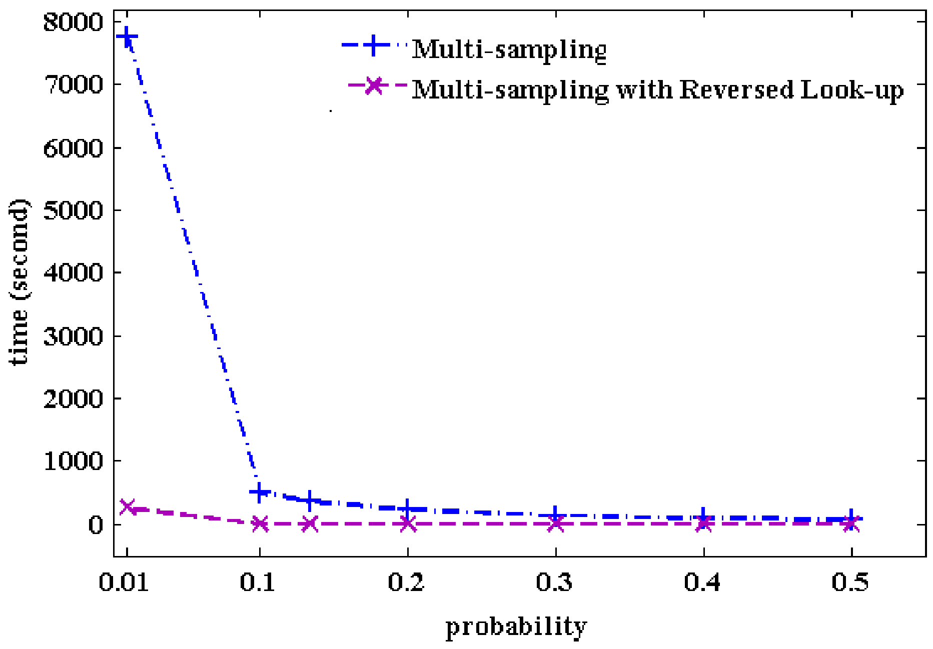

Figure 5.7: CPU time for the Multi-sampling variants ($N = 100,000$)

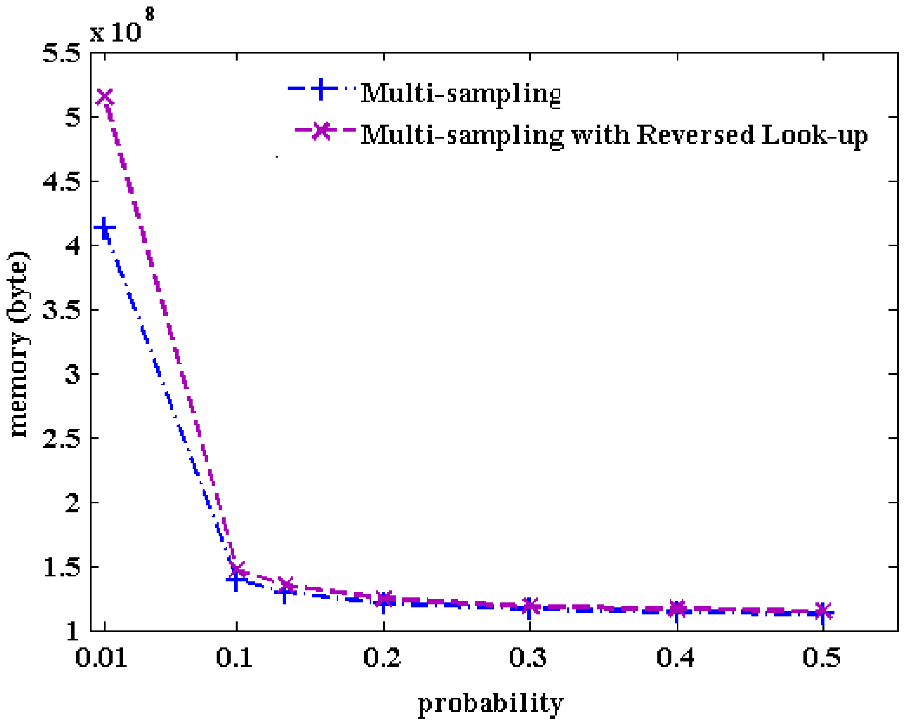

Figure 5.8: Memory use for the Multi-sampling variants ($N = 100,000$)

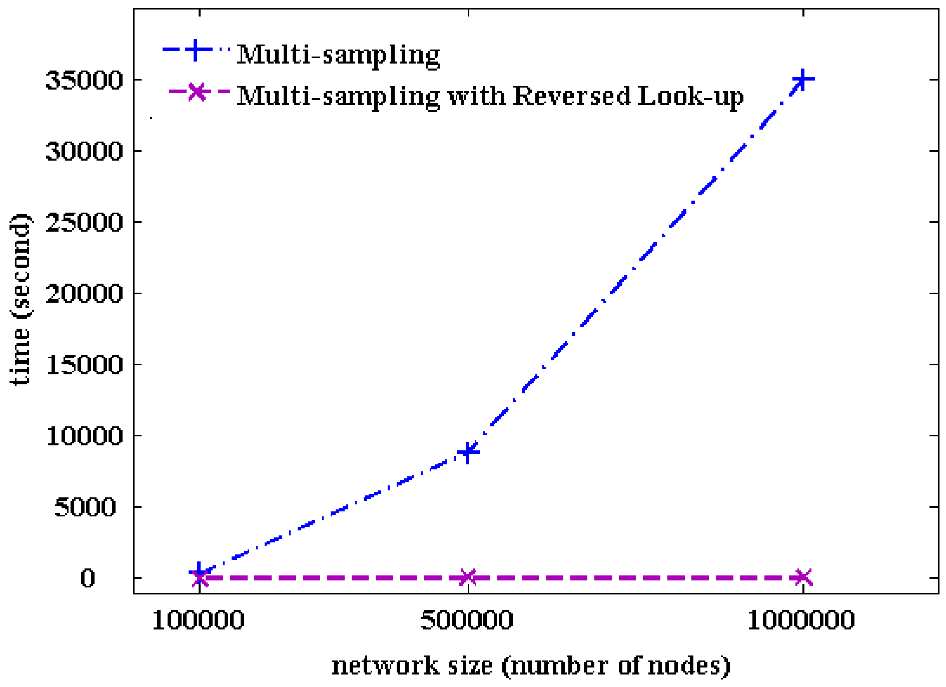

Figure 5.9: CPU time for large networks ($p = 0.1333$)

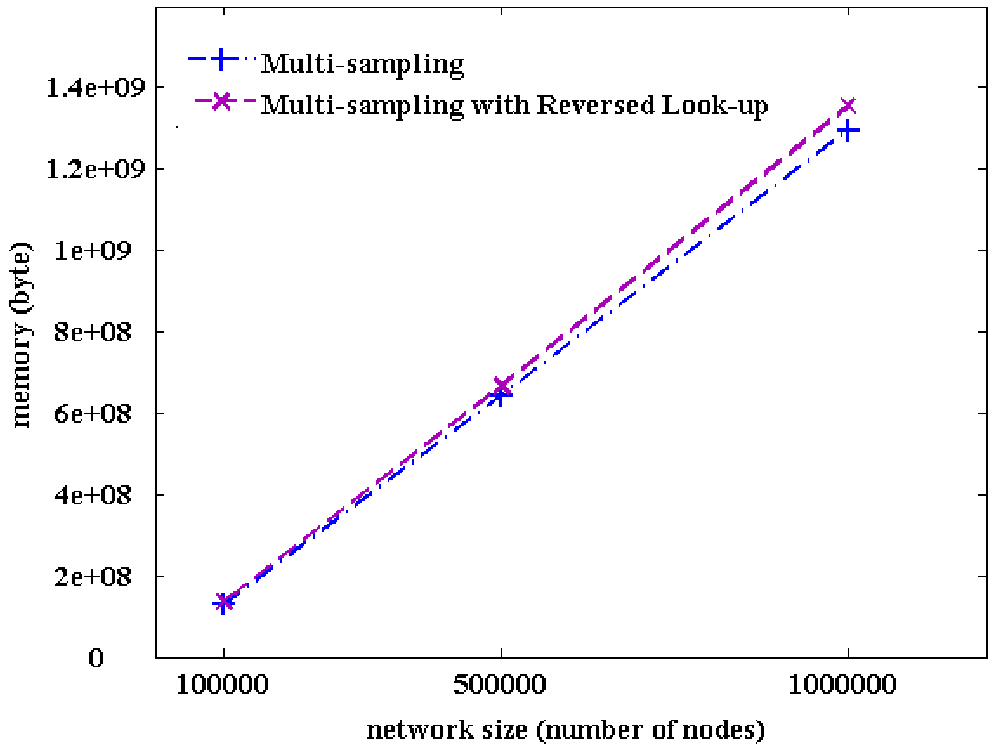

Figure 5.10: Memory use for large networks ($p = 0.1333$)

5.9). Considering memory usage, Figure 5.10 shows that the two methods do not differ too much. Moreover, with 1 million nodes we are exactly at the limit of the available memory of about 1.4GB. In other words, Multi-sampling with Reversed Look-up is the preferred approach, allowing us to generate networks with up to one million nodes in seconds (to be precise, 32 seconds for 1 million nodes).

5.5 Summary

This chapter is particularly focused on the development of fast algorithms that allow the model to be effectively used in discrete-event simulation studies. In Section 5.2 we briefly recap the network algorithms and introduced the related problems. In Section 5.3, We have derived an approach termed Multi-sampling with Reversed Look-up that under almost all circumstances outperforms other methods. The experiment results in Section 5.4 show that the amount of re-sampling required in the method is bounded and does not significantly reduce the applicability of the method across a broad parameter range. In addition, although the method requires additional memory to speed up the process of weighted random sampling, memory use does not much exceed that for maintaining the network itself (necessary for all algorithms). As a consequence, using Multi-sampling with Reversed Look-up one can generate networks with a million or more nodes within seconds on current-day desktops.

Chapter 6

Evaluation of P2P Algorithms for Probabilistic Trust Inference

6.1 Introduction

The previous chapter describes several algorithmic implementations and presents a quantitative comparison of their performance in theory, as well by simulation. This is a basic contribution to build the underlying network of our trust model. Let us recap our trust model here. Our trust model views the system similar to Web of Trust, a network or graph where nodes are linked if they have a trust relation. We assume links are directed, that is, a link or arc from A to B implies that A trusts B, but not B does not necessarily trust A. The problem we address is if, for a given pair of request node and target node, a trust path exists in the trust network. We associate a probability with each link to represent the trust value associated to the trust relation (either specified by the requester or by the trusting party associated with an arc). The overall trust value of a trust path is the product of the probabilities on the links. Moreover, when multiple trust paths exist between requester and target, the problem of computing the overall trust value translates to the network reliability problem, as pointed out in (Jonczy et al., 2006).

Given the above trust model, we are interested in evaluating how peer-to-peer algorithms perform when used for identifying trust paths. We quantify their performance by comparing the overhead (in number of messages used) with the achieved success rate (in fraction of paths found). We then also compare the achieved trust value with the trust value obtained if all trust paths are considered. In all steps of this study we use Monte Carlo and discrete-event simulation: for generating the networks, executing the peer-to-peer algorithms, and sampling the resulting paths to obtain the trust value. This chapter builds on the earlier work in the same area (Ribeiro de Mello et al., 2007), but that work was limited to the question if at least one trust path could be found, thus not including the overall success rates, nor introducing trust values and trust value computation.

The rest of the chapter is organized as follows. Section 6.1 provides an overview of the problems and analysis and explore the potential solutions. The analytical search algorithm technique are presented in Section 6.2. Numerical experiments are detailed in Section 6.3. The performance and cost comparison among various P2P algorithms to obtain probabilistic trust inference in Section 6.4. Section 6.5 gives a summary of the chapter.

6.2 Problem Definition

6.2.1 Trust path discovery problem

In abstract terms, one can model a web of trust as a directed graph $G = (V, E)$, in which the set of vertices V represent the nodes and the set of directed edges E represent trust relations. A directed edge from i to j corresponds to i's trust in j. In terms of this trust relation, i is the truster (Jøsang et al., 2006), and j is the trustee (Jøsang et al., 2006). As an example, in the PGP trust model (Jonczy et al., 2006), vertices are public keys and edges are certificates. A directed edge from i to j represents i signing a certificate which binds j to a key.

Assume trust relations exist between some of the nodes of a network, and based on direct interactions between them, direct trust (Mahoney et al., 2005) is created. But since not all nodes of a network have direct interactions, direct trust links do not exist between all pairs. Nodes without direct interactions, however, can estimate trust depending on its trustees and so on. For instance, as in (Ribeiro de Mello et al., 2007), assume there is no directed edge from A to C, A can still trust C if there exists at least one path from A to C in the graph. That is, it is accepted that trust is transitive in the trust model: if A trusts B, and B trusts C, then A trusts C. The trust A places on C is viewed as indirect trust (Mahoney et al., 2005), which is derived from the beliefs of others. As pointed out in (Mahoney et al., 2005), trust relations usually are one-way: A trusts B does not mean B trusts A.

In our model, every node maintains its trust relations associated with other nodes. If a node (called the requester) wants to establish the trustworthiness of another node (called the target) with which he has had no direct interactions before, the trust path discovery problem is to find one or more trust path to the target. In (Ribeiro de Mello et al., 2007), the authors indicate that P2P search algorithms can be applied to discover trust paths, due to the similarity of the structure of a web of trust network and an unstructured P2P network. In what follows, the process of finding trust paths is called a search phase. After the search phase, and if such trust path exists, the requester requires some mathematical method to estimate the trustworthiness of the target. How to develop a reasonable trust measure is presented in the following subsection. On the other hand, if no such trust path exists, the trustworthiness of a target cannot be established from the requester's point of view.

6.2.2 Trust inference problem

How to define a reasonable trust metric to estimate the trust placed on the target? Our work follows the approach in (Jonczy et al., 2006), which shows that the trust inference problem can be translated into the two-terminal network reliability problem.

Network reliability concerns the performance study of computer networks, in which components are subject to random failures. In network reliability analysis, it is assumed that edges have probabilities of failure (Ball et al., 1995). That is, at a particular time, an edge can take one of the two states, operative or failed. The two-terminal network reliability is computed as a probability of establishing at least one operating network path from s to t. Mapping the (s, t) network reliability problem to web of trust context, requester-target trust inference is viewed as determining the probability of establishing at least one trust path from the requester to the target.

To solve the network reliability problem, exact methods and approximate methods have been developed. In general, exact methods first calculate minimal operating states, path sets or cut sets, and then apply inclusion-exclusion principle to compute the result (Ball et al., 1995). However, exact methods suffer from an exponential worst-case complicity (Ball et al., 1995). That is caused by the computation of path sets or cut sets, which is an NP-hard problem (Ball et al., 1995). A Monte-Carlo technique belonging to approximate methods is commonly proposed and employed in network reliability computation (Fishman, 1995). This method is implemented in our experiments to compute the trust value.

6.3 Methodology

6.3.1 Topologies

The topology of a P2P network may influence the effectiveness of various search algorithms. We focus on two network topologies in our study: random graph and scale-free graph. Random graph is generated by the approach provided by Peersim (Jesi, 2004), while scale-free graph is generated by the approaches which were presented in the previous sections.

Random Graph. Given the network size S (the number of nodes) and an integer value d, PeerSim generates randomly d directed links out of each node. The neighbours are chosen randomly without replacement from the nodes of the network except the source node. We modified the basic algorithm in Peersim so that the out-degree follows a truncated standard normal distribution around d.

Scale-free Graph. We use network generating algorithm KRRNoLM (referred to Chapter 3) to generate directed scale-free networks without multiple arcs and self loops. The fastest algorithms (referred to Chapter 5) of generating the desired networks are implemented in Peersim. A set of parameters (S, p, λ, μ) needs to be specified as the input of a simulation.

6.3.2 P2P search algorithms

The trust path discovery algorithms considered in this chapter all are variations of flooding in unstructured P2P networks. More specifically, they are controlled flooding algorithms. For these approaches in the context of file sharing, we refer to (Lv et al., 2002).

Flooding. "Pure" Flooding has been mainly used in Gnutella networks (Gnutella, 2001). In this approach, a requester sends query messages to every node to which it directly connects. Receiving a query message, if a node does not find information being searched, it will forward this query message to all of its connected neighbour nodes. To avoid unlimitedly propagating messages, every query message is fixed with a time-to-live (TTL) parameter, which takes a positive integer value. Each time the query is forwarded by a node, the TTL value is decremented by 1. When the TTL value reaches zero, the query message will stop to be forwarded. We will see later that setting the TTL value is a critical aspect for the performance of the algorithm.

Random querying. In comparsion with Flooding, in Random querying, a requester sends query messages to a subset of its neighbour nodes, which is set to K percent of its neighbour nodes rounded below or rounded above. Upon receiving the incoming query, a node then continues forwarding the messages to its K percent randomly selected neighbour nodes. This method also relies on TTL parameter to limit the search.

Selective querying. Rather than forwarding incoming queries to randomly chosen neighbours, the Selective querying approach (Yang and Garcia-Molina, 2002) intelligently selects a subset of neighbours according to some specific criterion, for instance, the latency of connection, number of results for past queries, location and message queue capacity, etc. In the trust path searching, best neighbours are nodes with the most trust relations.

6.3.3 Metrics

To measure the efficiency of these algorithms, we considered three aspects reflecting the fundamental characteristics of the algorithms.

Success rate: the fraction of for which an algorithm successfully locates the target.

Number of messages: overhead of an algorithm is measured as the total number of search messages passed over the network during the search.

Trust inference value: A probability within a range $[0, 1]$, where 0 denotes no trust, and 1 denotes full trust. If no trust path exists, the trust value is 0, otherwise, the probabilistic trust value is computed as the solution of the network reliability problem as discussed in Section 6.2.2.

6.4 Simulation Methodology

In the following sections we show performance results of different P2P search algorithms described in the previous section. To better understand the impact of each algorithm in different scenarios, we used simulation, which allow us to evaluate their performance. In this section, we explain details of our simulations.

6.4.1 Peersim

We use PeerSim for our simulations. PeerSim is a Peer-to-Peer simulation framework, which is implemented in Java. It can be used to model any kind of P2P search algorithms. PeerSim simulator consists of several different components which can be easily plugged together by using an ASCII configuration file. It can work in two different modes: cycle-based or event-based. The cycle-based engine is a sequential simulation, in each cycle every node executes its own protocol's actions in a global sequential order. In the event-based mode, events are scheduled in different simulation time and nodes execute protocols according to message delivery times (Jesi, 2004). A very detailed account of performance and scalability comparison between these two modes is studied in (Defude, 2007). As recommended in (Defude, 2007), cycle-based mode of the PeerSim simulator is used in our study.

6.4.2 Sampling method

As a network topology consists of an infinite number of possible network instances, it is impossible to survey all its members to obtain the characteristics of a network topology. But a small cautiously chosen sample can be used to achieve the same aim.

In sampling technologies (Cochran, 1977), the term population denotes the complete set of observations that one wants information about, while the term sample stands for a subset of the population that we actually examine. In an experiment, a sample is selected from the population and statistic is collected from experimental samples in order to draw the conclusion about some properties of the population. In our simulation, a particular network topology (e.g. random network) is viewed as a population.

To simulate P2P search algorithms, the process of obtaining a sample is as follows: at first draw particular networks; then, within networks, select search queries (requester-target pairs). This way of selecting sample is called the Subsampling approach (Cochran, 1977). Sample selection is done in two steps: first select a sample of units from the population, named the primary units, and for each chosen primary unit, a sample of subunits are selected.

In the experiment, a particular network structure related to a specific network is viewed as a primary unit; one specific query inside is treated as a subunit. We use the symbol N to denote the population size, then a network topology consists of N primary units. Within a particular network (size S), if every node looks up all the other nodes, there will be a total of $S(S-1)$ query subunits constituting a particular network unit.

The following notation is used for obtaining estimate sample means and variances in Subsampling (Cochran, 1977):

n : number of primary unit samples

N : the total number of primary units

m : number of subunit samples per unit

M : the total number of subunits

$y_{i,j}$: value obtained for the jth subunit in the ith primary unit

$\bar{y}_i = \frac{1}{m}\sum_{j=1}^{m} y_{i,j}$ = sample mean per subunit in the ith primary unit

$\bar{\bar{y}} = \frac{1}{n}\sum_{i=1}^{n} \bar{y}_i$ = over-all sample mean per subunit

$f_1 = \frac{n}{N}$ = ratio of the size of the sample to the total of the primary units

$f_2 = \frac{m}{M}$ = ratio of the size of the sample to the total of the subunits

$s_1^2 = \frac{\sum_{i=1}^{n}(\bar{y}_i - \bar{\bar{y}})^2}{n-1}$ = variance among primary unit means

$s_2^2 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{i,j} - \bar{y}_i)^2}{n(m-1)}$ = variance among subunits within primary units

$v(\bar{\bar{y}}) = \frac{1-f_1}{n}s_1^2 + \frac{f_1(1-f_2)}{mn}s_2^2$ = sampling variance

In sampling, sampling variance can be calculated to show the degree to which a sample may differ from the population. As the total number of members in the population N is infinite in our experiments, $f_1 = \frac{n}{N}$ is negligible, and then we obtain that the estimated variance can be computed as:

$$v(\bar{\bar{y}}) = \frac{s_1^2}{n} = \frac{\sum_{i=1}^{n}(\bar{y}_i - \bar{\bar{y}})^2}{n(n-1)} \quad ,$$

and the estimated sample standard deviation is $s_(\bar{\bar{y}}) = \sqrt{v(\bar{\bar{y}})}$.

Given the estimated sample mean and sample standard deviation, if t_c is the t value associated with $c\%$, then a $c\%$ confidence interval for the mean is equal to $y \pm t_c s(\bar{y})$. For instance, if the desired confidence probability is 95%, the t_c value is 1.96. Then we say that a 95% chance that the population mean is within a range of $[\bar{y} - t_c s(\bar{y}), \bar{y} + t_c s(\bar{y})]$.

Statistics are collected from the n (number of primary unit samples) $\times$ m (number of subunit samples) queries. For the result we present in the paper, n=50 and m=50, which turns out to ensure small standard deviation in our results.

6.4.3 Trust computation

Given a pair of requester and target, if there is at least one trust path exists, the trust value is computed based on the trust paths discovered. The overall trust value of a trust path is the product of the probabilities on the links. Moreover, when multiple trust paths exist between requester and target, the problem of computing the overall trust value translates to the network reliability problem. A simple example is given below to demonstrate how the trust is computed when multiple trust paths exist.

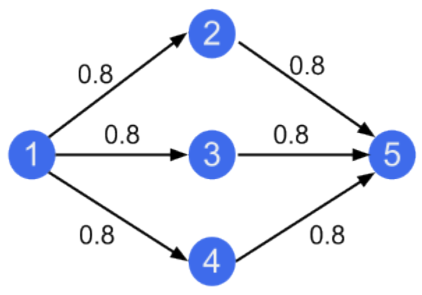

Figure 6.1: Example multiple trust paths exist between requester node 1 and target node 5

In Figure 6.1, there are 3 independent trust paths exist between requester node 1 and target node 5. The overall trust value is then computed as the probability of at least one trust path existing, in other words, $1-$ the chance of no trust path existing. We assume that each arc has the same probability value 0.8, therefore the overall trust value is $1-(1-0.8*0.8)^3$.

As explained in Section 6.2.2, in our experiments, we implement the Monte-Carlo method. The Monte-Carlo method is a computation which performs statistical sampling to obtain the result (Fishman, 1995). As a consequence, the trust computation effectively becomes 'three-level-unit' samples. The primary units are the drawn particular networks, the secondary units are chosen random queries, and finally the tertiary units are generated trust graph samples. As a consequence, trust mean is:

$$\bar{\bar{\bar{y}}} = \frac{1}{n}\sum_{i=1}^{n}\{\frac{1}{m}\sum_{j=1}^{m}(\frac{1}{l}\sum_{k=1}^{l}y_{i,j,k})\}.$$

For simplify, it is assumed that each edge has the same trust reliable value. In the experiment, the value was set to 0.8. Experiment results and discussions on interesting finding are presented in the following section.

6.5 Experimental Results

In this section, we start with the overview of the network topologies generated by the simulator, and then discuss performance results in random and scale-free networks, respectively. We assume that the trust network graph does not change during the simulation of the algorithms. Effectively, this implies that the time to complete a search enough so that no nodes leave or enter the network. We also assume testing the behaviour of a search algorithm in a stable environment, where no node failures were induced.

Network Characteristics. Before discussing the algorithm performance, we have a look at the networks on which the simulations perform. Figures 6.2 and 6.3 show the complementary cumulative distribution function for node in-degree and out-degree distributions of directed scale free networks with different p values. In log-log scale, the distributions visually look like a straight line.

Parameters Values. Our simulations were carried out in a network of size S=10000. P2P search algorithms applications are tested on two types of network topology: random and scale-free topology. In the random networks, the average out-degree value is 7. The scale-free networks are drawn with three different p values (0.05, 0.1333, 0.5), where p can be viewed as the speed of adding a new node to the network. For Random and Selective querying algorithms, we chose three different values (10%, 50%, 70%) for the fraction of neighbours to which each query will be forwarded.

One can be interested in how the nodes link to each other. If there is at least one path leading to node j from node i, then we say this node pair $<i,j>$ is connected. We use the term node pair connection ratio (connection ratio for short) to present the fraction of node pairs being connected in a network. The node pair connection ratio is strongly influenced by network topology and the p value among scale free networks, which can be seen from Table 6.1. The query samples are the secondary units, and the network samples are the primary units. In the random networks, both in the query samples and network samples, the node pair connection ratio is over 99%, which implies nearly every node is connected to all the others. On the other hand, in the scale-free networks, the connection ratio is much smaller, although it increases with lower p value. The reason why the connection ratio is smaller is that the number of nodes with zero

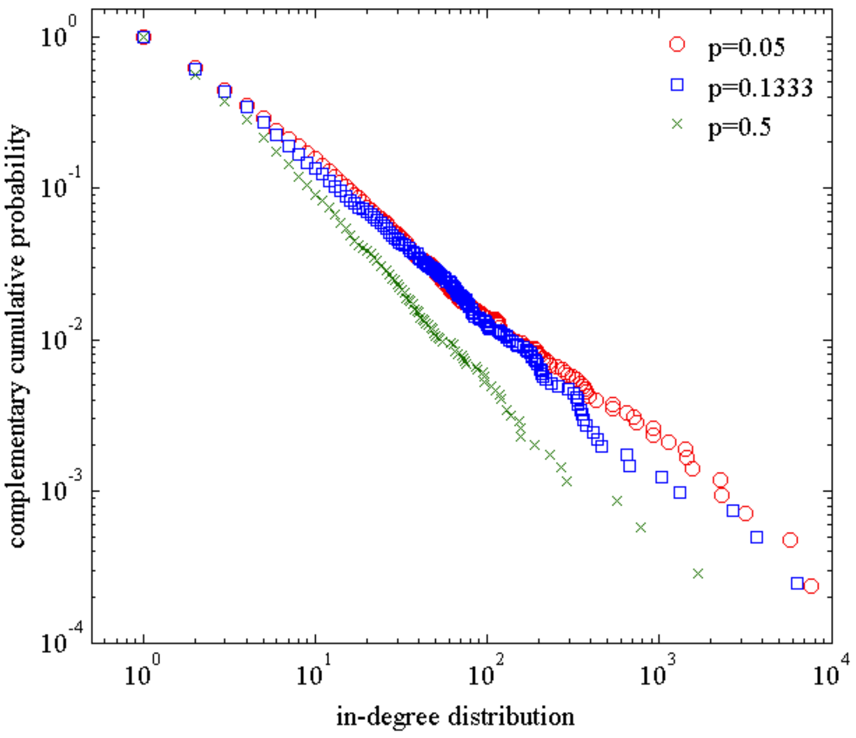

Figure 6.2: The average complementary cumulative node in-degree distribution of directed scale free networks generated with various p, versus the in-degree of the networks with 10,000 nodes

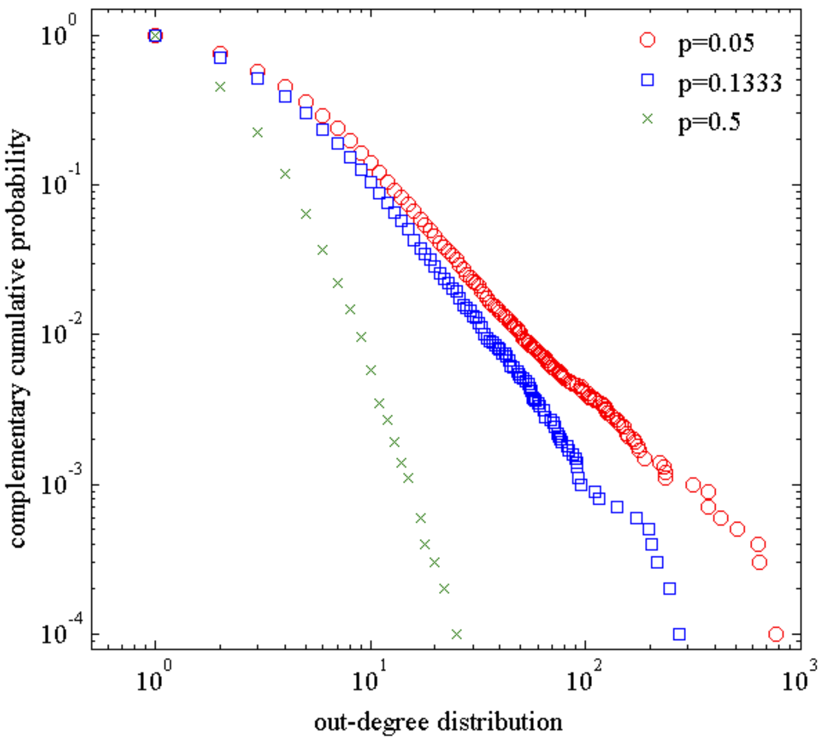

Figure 6.3: The average complementary cumulative node out-degree distribution of directed scale free networks generated with various p, versus the out-degree of the networks with 10,000 nodes

Topology	Node Pair Connection Ratio, %	
	query samples	network samples
Random network		
out-degree=7	99.76	99.89
Scale-free network		
$p = 0.5$	14.20	13.00
$p = 0.1333$	31.84	32.19
$p = 0.05$	38.96	38.26

Table 6.1: Node pair connection ratio in random and scale-free networks

in-degree value is large. That means many nodes are not reachable from other nodes, resulting in a low success rate. To obtain a higher success rate, we look at more possible higher connection ratios by given smaller p values (to achieve higher average out-degree values). For random networks, the connection ratio is satisfactory. It can also be seen from Table 6.1, that the connection ratio in the query samples is similar to that in the network samples, which means the query samples basically reflect the feature of the node pair connection in the network samples.

6.5.1 Results in random networks

Figures 6.4 and 6.5 present messages overhead and probability of success as TTL increases in random network. The shown lines are in the order they appear in the graphs. It can be observed that Flooding always has higher overhead and higher success rate than all the others, for identical TTL values. Random querying (70%) and Selective querying (70%) achieve similar success rate, quite a bit smaller than flooding until TTL reaches 5. From TTL=6 onwards, both algorithms obtain similar success rate close to that of Flooding.

The key insight gained from our study is given in Figure 6.6, which combines the results of the two Figures 6.4 and 6.5. It shows for each algorithm the message overhead versus the success rate, and each curve consists of eight points, with the results for TTL=2 until 9. It can be seen from Figure 6.6, that it does not matter if one uses Flooding or Random querying/Selective querying. For any success rate, the message overhead of the algorithms is similar. This

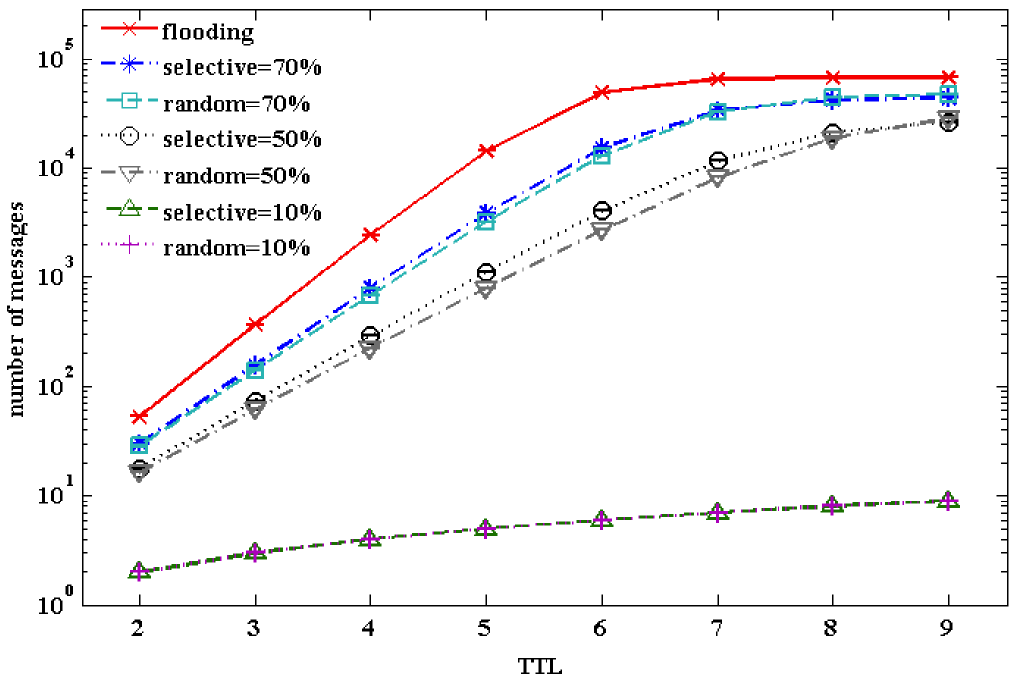

Figure 6.4: Message overhead for different TTL values in random network (lines are in the order they appear in the graphs)

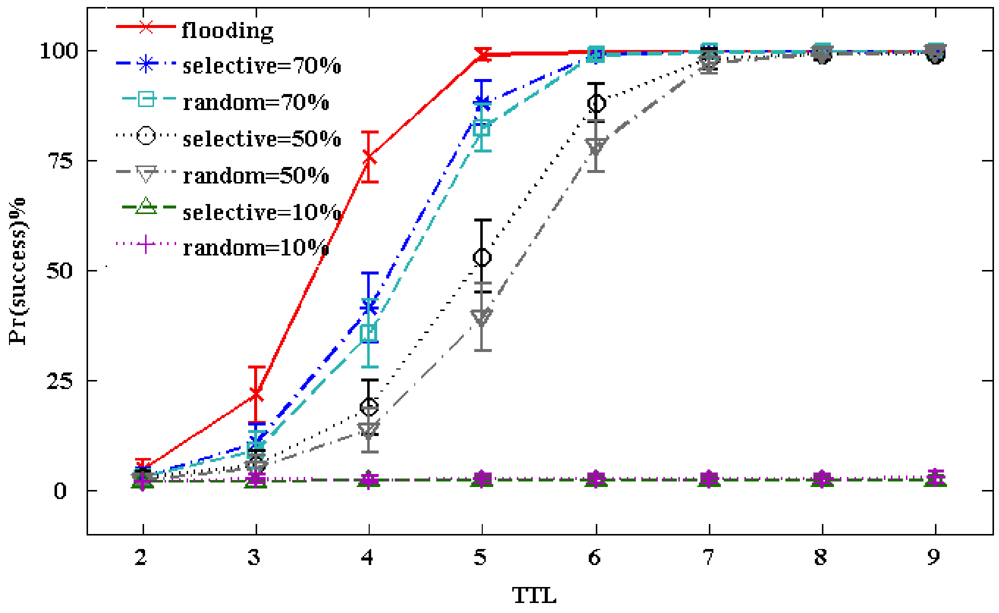

Figure 6.5: Success probability for different TTL values in random network (lines are in the order they appear in the graphs)

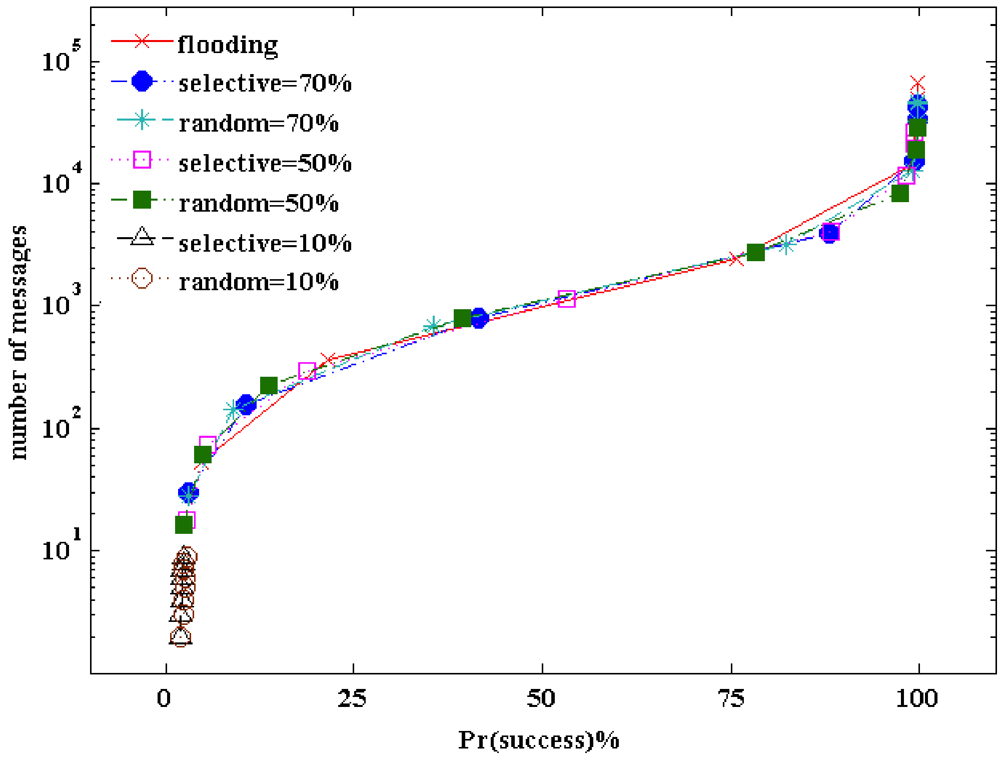

Figure 6.6: Message overhead versus success rate in random network

implies that for a given algorithm, if one knows the TTL value that achieves the desired success rate on message overhead, this algorithm will be close to optimal. The problem is, of course, that the correct TTL value is not known beforehand. Note furthermore that Flooding achieves the highest success rate (as one would expect), but that the Selective/Random algorithm is competitive if the percentage is set high enough (70% in our case). For lower percentage, even very large TTL values may not provide the success rate achievable with Flooding. This indicates a second complication in Selective/Random querying: the percentage must be set, and the optimal value is (like in the case of the TTL value) not known.

The exact trust value would be obtained if all trust paths would be considered. As a consequence, all results in Table 6.2 are lower bounds for the trust value. Since Flooding has the highest success probability, it is not surprising that it also obtains the highest trust values. In particular, we see that the trust value of random networks is at least 0.991. One also see from Table 6.2 that a TTL value of at least 7 is needed for Selection/Random querying to give satisfactory results, and that the percentage must be set to at least 50%.

P2P Search Algorithm	Trust Inference			
	TTL=6	TTL=7	TTL=8	TTL=9
Flooding	0.991	0.991	0.991	0.991
Random querying				
10%	0	0.001	0.001	0.001
50%	0.388	0.845	0.968	0.981
70%	0.948	0.988	0.990	0.990
Selective querying				
10%	0	0.001	0.001	0.001
50%	0.630	0.923	0.964	0.969
70%	0.961	0.986	0.987	0.988

Table 6.2: Probabilistic trust inference values in random networks

6.5.2 Results in scale-free networks

We consider scale-free networks with three different p values (0.5, 0.1333, 0.05).

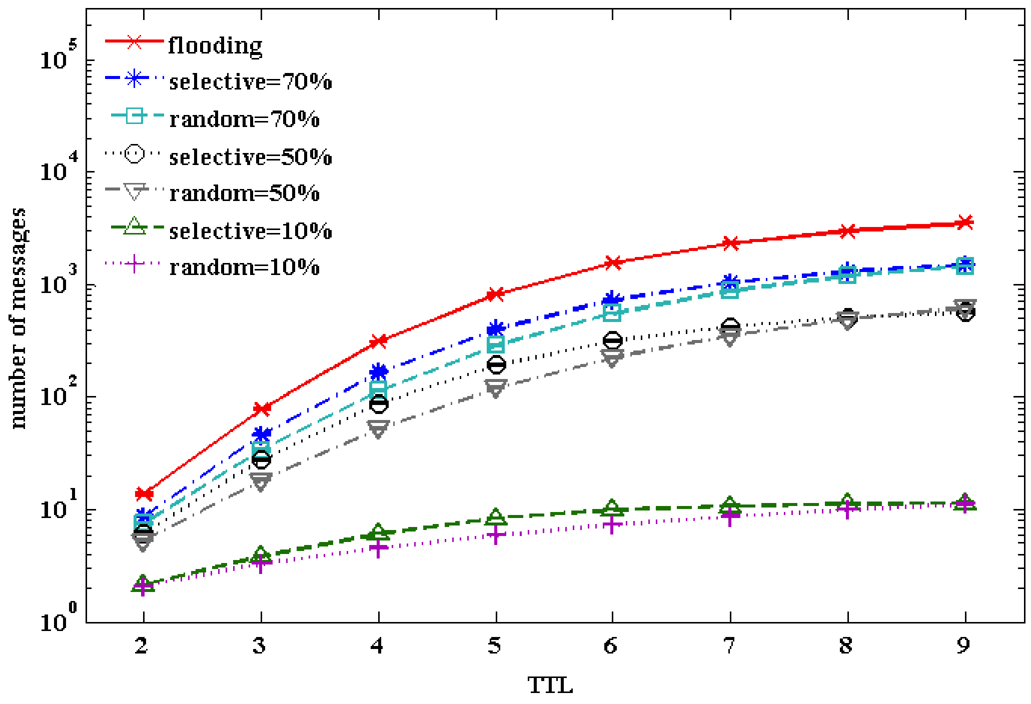

Figure 6.7: Message overhead under various TTLs in scale-free networks ($p = 0.5$)

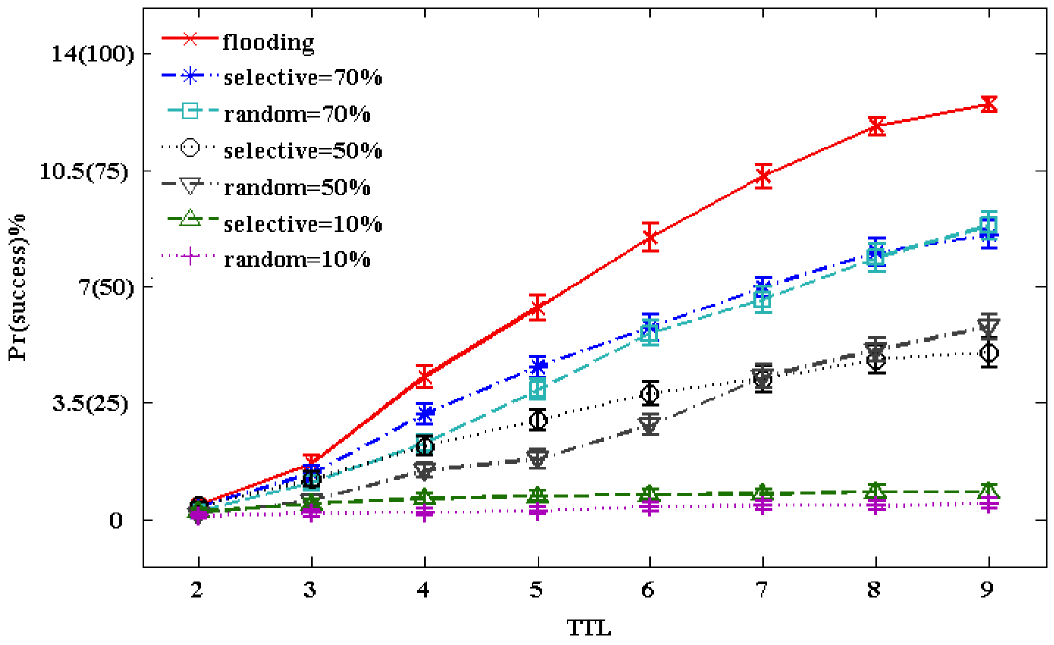

Figure 6.8: Success probability under various TTLs in scale-free networks ($p = 0.5$)

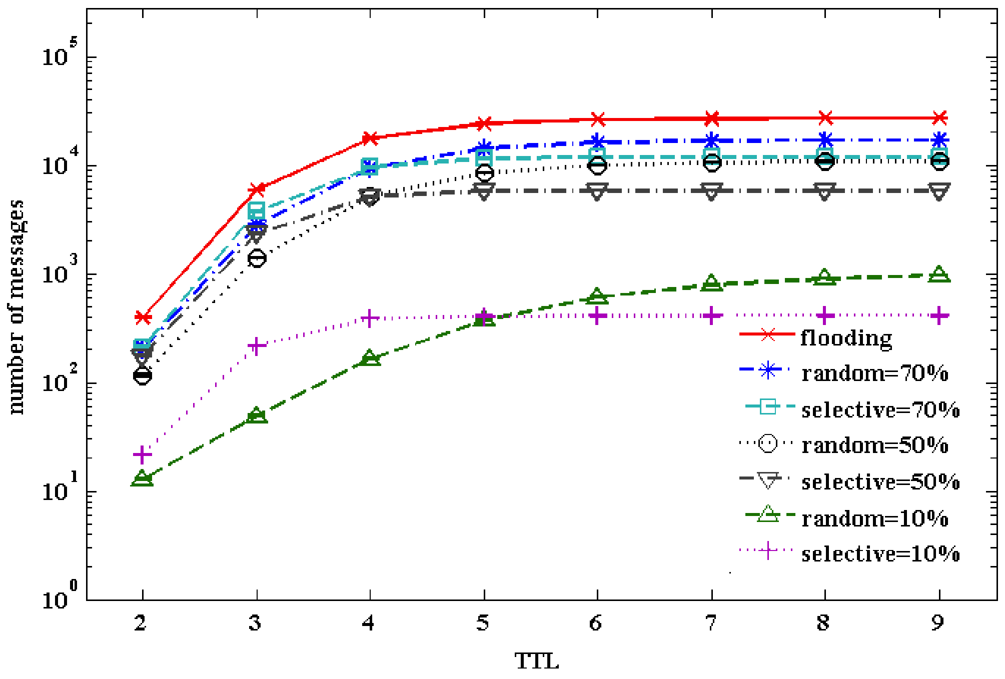

Figure 6.9: Message overhead under various TTLs in scale-free networks ($p = 0.1333$)

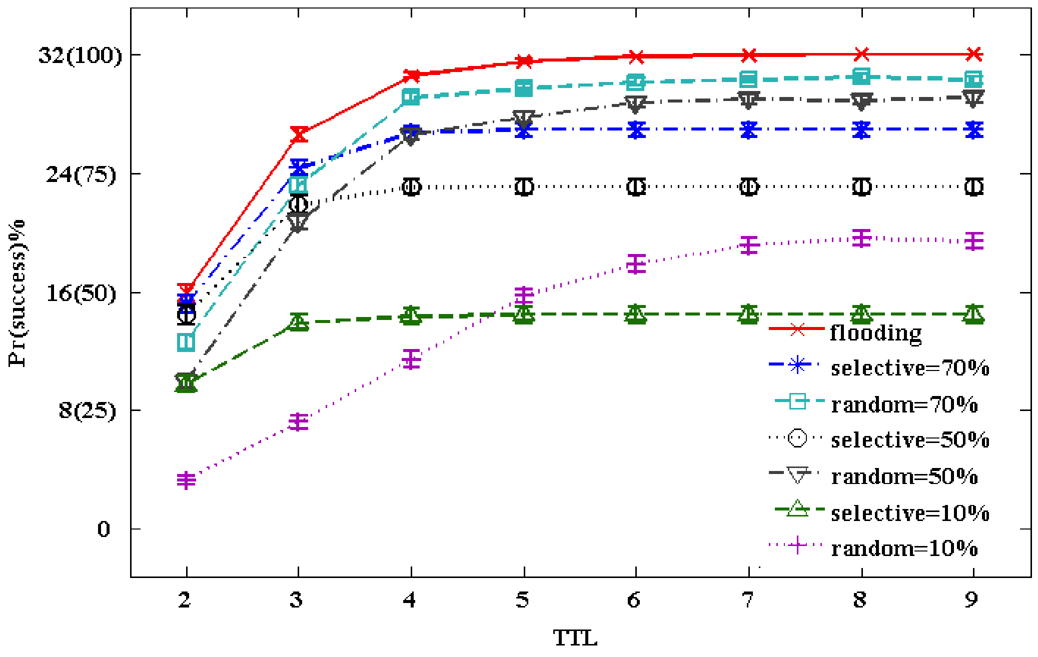

Figure 6.10: Success probability under various TTLs in scale-free networks ($p = 0.1333$)

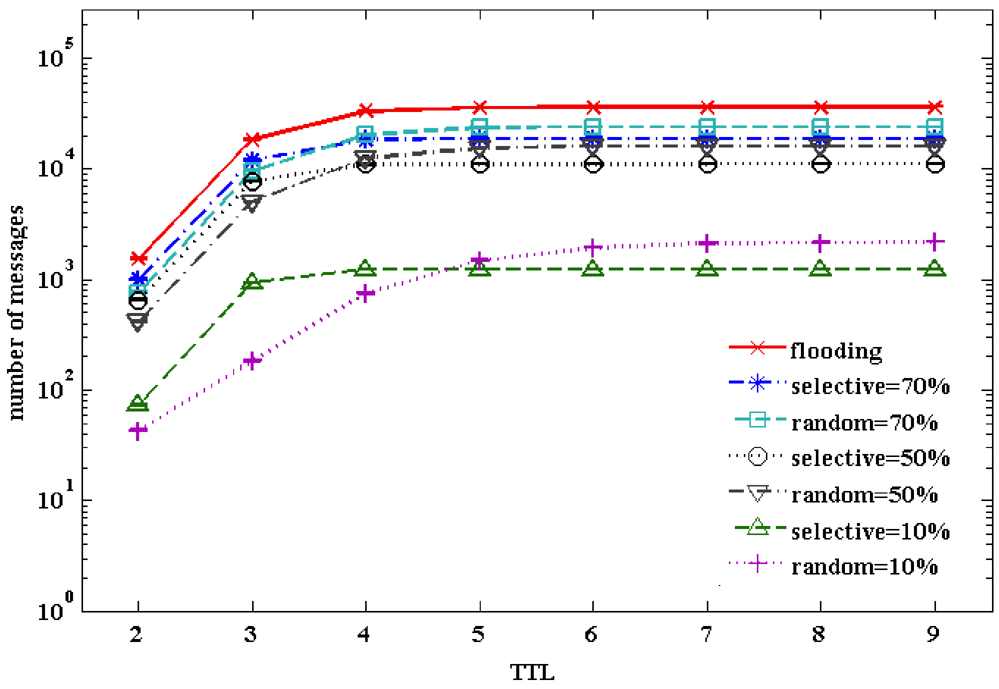

Figure 6.11: Message overhead under various TTLs in scale-free networks ($p = 0.05$)

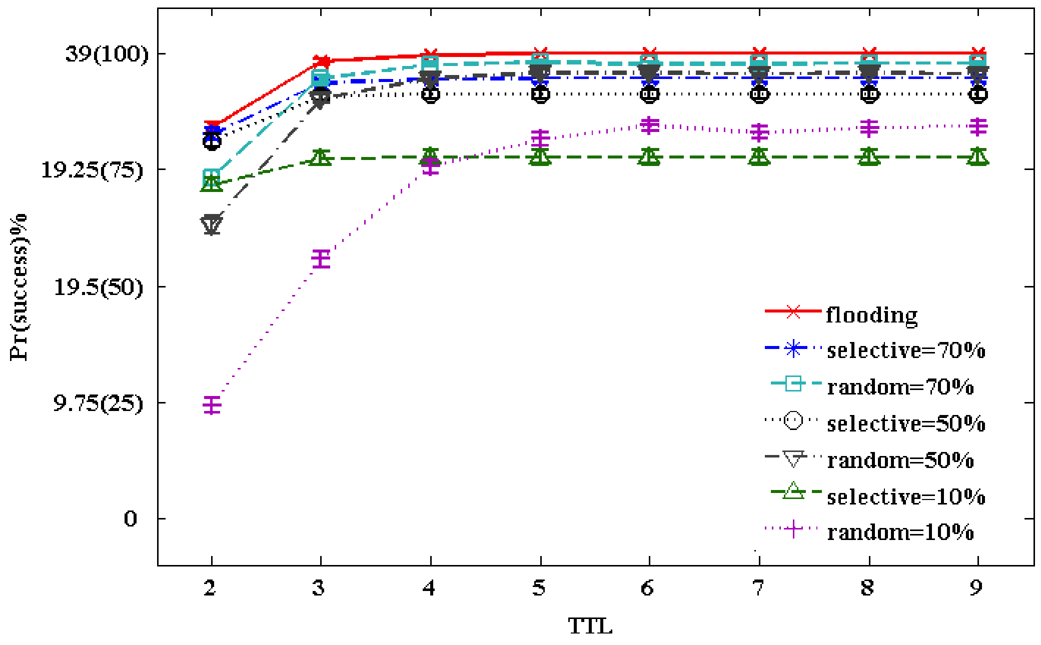

Figure 6.12: Success probability under various TTLs in scale-free networks ($p = 0.05$)

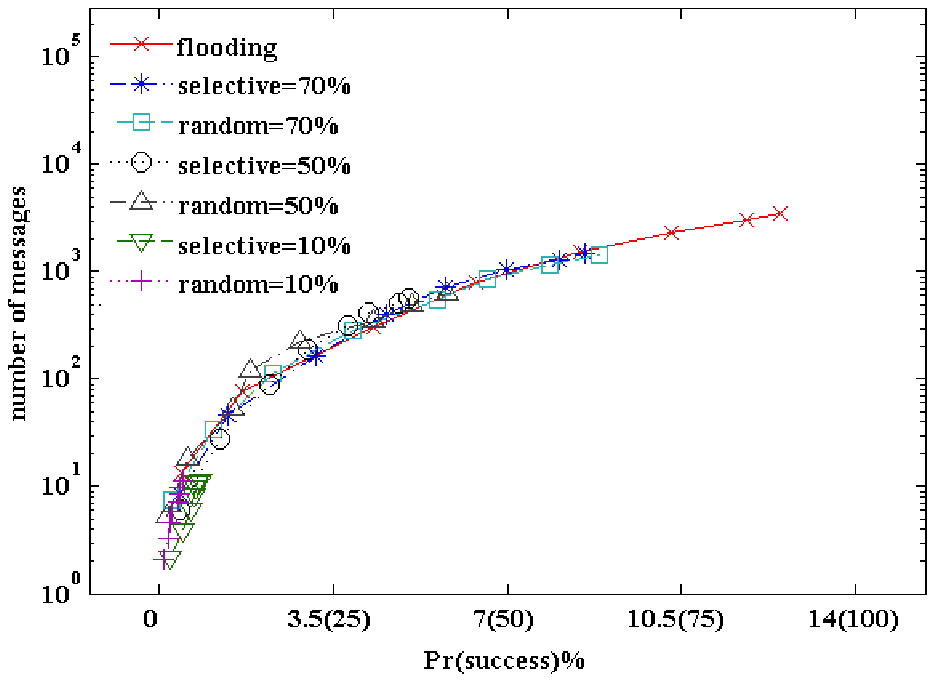

Figure 6.13: Message overhead versus success rate in scale-free networks ($p = 0.5$)

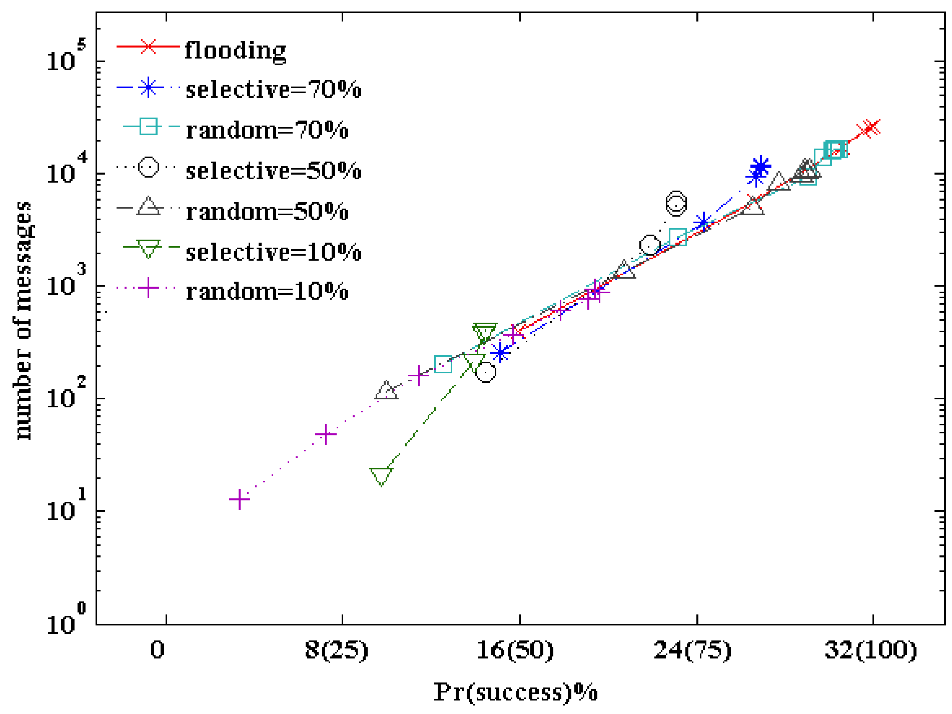

Figure 6.14: Message overhead versus success rate in scale-free networks ($p = 0.1333$)

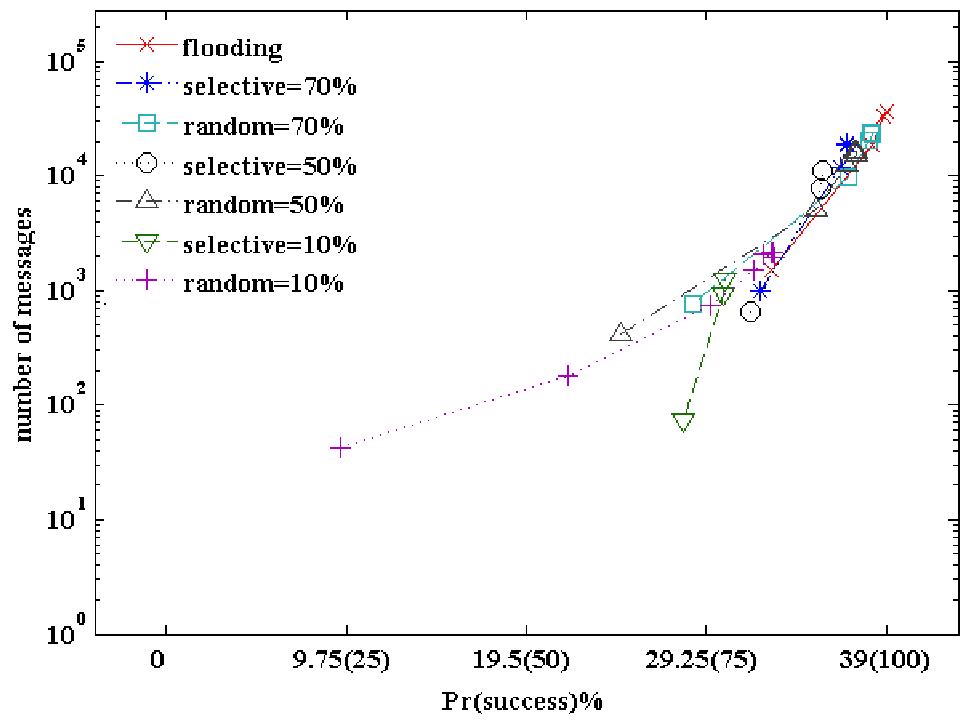

Figure 6.15: Message overhead versus success rate in scale-free networks ($p = 0.05$)

Figures 6.7, 6.9, 6.11 show the number of messages propagated through scale-free networks with different p values (0.5, 0.1333, 0.05), for different values of TTL. Figures 6.8, 6.10, 6.12 present the success rate of searches. The y-axis of these figures gives the success probability as well as a normalised success probability between brackets. Since each algorithm finds only a subset of all trust paths, the success probability of an algorithm is bounded by the percentage of node pairs in a network for which a trust path exists. That maximum value is given on the y-axis with 100 between brackets. The percentage between brackets is thus a normalised success probability. For instance, in Figure 6.8 the success probability of the network is 14%, and the flooding algorithm finds most existing trust paths for high values of TTL. The shown lines are in the order they appear in the graphs. Similar to random networks, for each TTL value Flooding has the highest overhead and highest success rate. We can see from Figure 6.10 that flooding algorithm finds almost all the trust paths when TTL reaches 6. In more connected scale free networks ($p = 0.05$), seen from Figure 6.12, Flooding algorithm finds almost all trust paths when TTL arises to 4.

As we can see from Figures 6.10 and 6.12, Selective querying achieves a little higher success rate than Random querying with the same K percent value until TTL reaches 4. From TTL=5 upward, Selective querying could not achieve any higher success rate, while Random querying keeps achieving a higher success rate. Now we will explain why Selective querying performs better than Random querying for small TTL but opposite is true for large TTL. Firstly, note that in the networks with smaller p values, nodes are more connected to other nodes. In Selective querying, nodes with more trust relations are chosen to forward query messages. Therefore, for the same small TTL, Selective querying is more likely to find target nodes. Note that we assume each node keeps track of every query message seen and only forward each query message once. In scale-free networks, high degree nodes tend to connect to other high-degree nodes. When TTL gets large, selective querying could not achieve any higher success rate, as query messages are forwarded to the nodes with more trust relations, which likely have already forwarded the same query messages. In Random querying, nodes are randomly selected to forward query messages. Therefore, Random querying can achieve a higher success rate with large TTL. As for random networks, we plotted success probability versus message overhead in Figures 6.13, 6.14, 6.15, we see that Flooding, Random querying(70%) and Selective querying(70%) perform similarly, but importantly Flooding can obtain a higher success rate.

Arguably, the difference between Flooding and other algorithms is even more pronounced in scale-free networks than in random networks. Note again that in the Selective/Random algorithms a high enough value for K (the percentage of selected nodes) must be chosen to achieve a reasonable success rate.

Table 6.3 shows the computed trust values. As we can see, in scale-free networks, the trust value is very low for all the search algorithms. This is caused by the low node pair connection ratio of scale-free networks, see Table 6.1. With the decrease of p, which increase the average out-degree value, the node pair connection ratio increases and therefore, trust increases. Flooding obtains the highest values, as can be expected, resulting in a lower bound of the trust value of 0.023, 0.094 and 0.206 for the respective p values. Random querying slightly outperforms Selective querying, even for the case of $p = 0.5$, in which Selective querying achieved a higher success rate.

6.5.3 Discussion

For both network topologies, we notice the amount of messages to obtain a high success rate is very sensitive to the value of K, the number of nodes to which a query is forwarded.

The main challenge in using any of the studied algorithms is to set the value of TTL. To improve the performance, we need to consider how to assign the TTL value when a search algorithm is initialized, and how to efficiently control or avoid unnecessary messages being forwarded when the target has been located.

To avoid excessive messages being forwarded, adaptive termination can be considered. When a trust path is located, the requester broadcasts "stop searching" messages to other nodes to terminate the search process by dropping query messages whose TTL does not reach 0 yet. In terms of message overhead, the Expanding Ring search algorithm (Lv et al., 2002) can be a potential solution. The Expanding Ring algorithm starts searching with a small TTL value. When TTL reaches 0 and the search is not completed, the TTL value is incremented by 1 and the search is continued.

Perhaps surprising (since it contradicts a possible tendency to think that flooding is expensive), our results show that these algorithms are largely equivalent when

P2P Search Algorithm	Trust Inference			
	TTL=6	TTL=7	TTL=8	TTL=9
$p = 0.5$				
Flooding	0.048	0.059	0.067	0.071
Random querying				
10%	0	0	0	0
50%	0.012	0.018	0.022	0.025
70%	0.028	0.033	0.038	0.043
Selective querying				
10%	0.002	0.002	0.002	0.002
50%	0.019	0.021	0.022	0.023
70%	0.030	0.036	0.040	0.042
$p = 0.1333$				
Flooding	0.241	0.242	0.242	0.242
Random querying				
10%	0.098	0.107	0.107	0.108
50%	0.201	0.204	0.203	0.205
70%	0.217	0.219	0.219	0.219
Selective querying				
10%	0.086	0.086	0.086	0.086
50%	0.163	0.163	0.163	0.163
70%	0.193	0.193	0.194	0.194
$p = 0.05$				
Flooding	0.312	0.311	0.312	0.312
Random querying				
10%	0.210	0.206	0.208	0.207
50%	0.281	0.281	0.281	0.281
70%	0.290	0.290	0.290	0.291
Selective querying				
10%	0.199	0.198	0.198	0.199
50%	0.265	0.267	0.267	0.268
70%	0.282	0.281	0.282	0.281

Table 6.3: Trust inference values for different algorithms in scale-free networks

considering the overhead versus the success rate, provided one sets the configuration parameters optimally. This holds true if the fraction of trust paths one wants to find is not too high. However, if a high success rate is required, flooding becomes superior, simply because it covers more paths. Selective and Random querying algorithms do not find all trust paths, and the resulting computed trust value is therefore lower than for pure flooding.

6.6 Summary

In this chapter, we used discrete event simulation and Monte Carlo techniques to evaluate the suitability of using peer-to-peer algorithms for discovering trust paths and inferring the trust value of a set of trust paths. The problems were defined in Section 6.2. In Section 6.3, we presented variations of the flooding search algorithm, and defined the metrics that would be used in the simulations. The simulation methodology including the sampling method and the trust computation method is explained in Section 6.4. From the results discussed in Section 6.5, our main conclusion is that all the variants of flooding perform almost equal when considering the message overhead for a certain probability of finding paths. When close to all paths need to be found, flooding outperforms selective flooding alternatives, since these alternatives miss out on certain paths.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis presents a study of algorithms to generate directed scale-free networks, their implementation in an efficient manner, and the performance of peer-to-peer search algorithms for trust paths.

To create directed scale-free networks without multiple arcs and self loops, we started with modifying an existing network generating algorithm, which was presented in Chapter 3. Based on this algorithm, we then developed growth and non-growth algorithms for generating the directed scale-free networks without multiple arcs and self loops in Section 3.3. We developed a mathematical generic framework for determining the conditions under which the degree distribution is power law. The mathematical generic framework shows that the growth algorithms are power law when the size of the network as well as in-degree and out-degree go to infinity.

To demonstrate the power law distribution of all the algorithms, an evaluation was carried out using a statistical analysis framework, which was given in Section 4.1. The experimental results in Section 4.3 show how the network size impacts the scale-free nature of the networks, taking into account various parameter values. Among all the algorithms we studied, only one algorithm consistently produces directed scale-free networks without multiple arcs or self loops. Unfortunately, this algorithm does not possess the growth nature of intuitively preferred algorithms. Statistical tests of the empirical data from OpenPGP was presented in Section 4.4, suggesting that the web of trust network exhibits a power law distribution for the node out-degree, but not in-degree.

The growth algorithms we proposed in Chapter 3 pose implementation challenges. In Chapter 5 we reported our experiences in developing efficient implementations. To reduce computation time and memory usage, we designed and implemented a set of algorithms, presented in Section 5.3. Both theoretical complexity analysis and experimental results in Section 5.4 show that the Multi-sampling with Reversed Look-up approach far outperforms the others. It generates networks with a million or more nodes within seconds on current-day desktops.

Finally, we use the developed approach to generate networks for discrete-event simulation studies of peer-to-peer search algorithms. An overview of problems and potential solutions were addressed in Section 6.2, variations of the flooding search algorithm were considered for the study of finding trust paths and probabilistically assessing trust values. P2P search algorithms were evaluated through simulation of random as well as scale-free networks. The performance results presented in Section 6.4 show that algorithms perform equally in many situations and only differ through (and are sensitive to) parameter choices.

7.2 Future Work

For further work, we would like to point out the following.

In Chapter 3, we presented a mathematical generic framework to establish conditions for growth algorithms to generate networks with the power law node degree distributions. However, we were not able to theoretically verify the conditions for our algorithms. In particular, it is difficult to know the probability of adding multiple arc or self loop. Therefore, additional formal analysis would be of interest to analyse if power law degree distributions are generated by the proposed growth algorithm.

In Chapter 6 variations of the flooding search algorithm have been studied for probabilistic trust inference. In Section 6.5.3, we addressed the challenge in using a class of flooding search algorithms. It would be of interest to study the performance of some improved(/alternative) peer-to-peer search algorithm(s).

To develope a network model to generate networks that have properties similar to the Web of Trust network, one can do the analyses in the other way. First get Web of Trust data. Next find the distribution of the data. Then design a model from that distribution. Or better yet, one can study the growth of Web of Trust over time and design a fitting model to generate similar networks.

Bibliography

Abdul-Rahman, A. (1997), "The PGP Trust Model", *EDI-Forum: The Journal of Electronic Commerce* , Vol. 10, pp. 27–31.

Aiello, W., Chung, F. and Lu, L. (2000), "A random graph model for power law graphs", *Experimental Math* , Vol. 10, pp. 53–66.

Anderson, S., Bohren, J., Boubez, T., Chanliau, M., Della-Libera, G., Dixon, B., Garg, P., Gudgin, M., Hallam-Baker, P., Hondo, M., Kaler, C., Lockhart, H. and Maruyama, H. (2005), Web Services Trust Language, Technical report.

Androutsellis-Theotokis, S. and Spinellis, D. (2004), "A survey of peer-to-peer content distribution technologies", *ACM Comput. Surv.* , Vol. 36, pp. 335–371.

Aringhieri, R., Damiani, E., Vimercati, S. D. C. D., Paraboschi, S. and Samarati, P. (2006), "Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems", *J. Am. Soc. Inf. Sci. Technol.* , Vol. 57, pp. 528–537.

Ball, M. O., Magnanti, T. L., Monma, C. L. and Nmehauser, G. L. (1995), *Network Models*, North Holland.

Barabasi, A.-L. and Albert, R. (1999), "Emergence of scaling in random networks", *Science* , Vol. 286, p. 509.

Berg, J., Lassig, M. and Wagner, A. (2004), "Structure and evolution of protein interaction networks: a statistical model for link dynamics and gene duplications", *BMC Evolutionary Biology* , Vol. 4, p. 51.

Bianconi, G. and Barabási, A.-L. (2001), "Bose-einstein condensation in complex networks", *Phys. Rev. Lett.* , Vol. 86, pp. 5632–5635.

Boeyen, S., Ellison, G., Karhuluoma, N., MacGregor, W., Madsen, P., Sengodan, S., Shinkar, S. and Thompson, P. (2004), *Trust Models Guidelines.*

Boguñá, M. and Pastor-Satorras, R. (2003), "Class of correlated random networks with hidden variables", *Phys. Rev. E* , Vol. 68, p. 036112.

Bollobás, B., Borgs, C., Chayes, J. and Riordan, O. (2003), Directed scale-free graphs, *in* 'SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms', pp. 132–139.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A. and Wiener, J. (2000), "Graph structure in the web", *Computer Networks* , Vol. 33, pp. 309–320.

Bulter, J. K. and Cantrell, R. S. (1984), "A behavioral decision theory approach to modeling dyadic trust in superiors and subordinates", *Psychological Reports* , Vol. 55.

Caldarelli, G. (2007), *Scale-Free Networks*, Oxford University Press.

Caldarelli, G., Capocci, A., De Los Rios, P. and Muñoz, M. A. (2002), "Scale-free networks from varying vertex intrinsic fitness", *Phys. Rev. Lett.* , Vol. 89, p. 258702.

Callas, J., Donnerhacke, L., Finney, H., Shaw, D. and Thayer, R. (2007), RFC 4880 - OpenPGP Message Format, Technical report, Internet Engineering Task Force.

Capocci, A., Servedio, V. D. P., Colaiori, F., Buriol, L. S., Donato, D., Leonardi, S. and Caldarelli, G. (2006), "Preferential attachment in the growth of social networks: The internet encyclopedia wikipedia", *Phys. Rev. E* , Vol. 74, American Physical Society, p. 036116.

Carbone, M., Nielsen, M. and Sassone, V. (2003), A Formal Model for Trust in Dynamic Networks, *in* 'In Proc. of International Conference on Software Engineering and Formal Methods (SEFM'03', pp. 54–63.

Cederlof, J. (2007), http://www.lysator.liu.se/~jc/wotsap/index.html. last visited on 10/10/2011.

Clauset, A. (n.d.), http://tuvalu.santafe.edu/~aaronc/powerlaws/.

Clauset, A., Shalizi, C. R. and Newman, M. E. J. (2009), "Power-law distributions in empirical data", *SIAM Review* , Vol. 51, p. 661.

Cochran, W. (1977), *Sampling Techniques*, 3rd edn, Wiley and Sons.

Coleman, J. S. (1990), *Foundations of Social Theory*, The Belknap Press of Harvard University Press.

Defude, B. (2007), ‘P2P simulation with peersim’, http://stromboli3.int-edu.eu/~bernard/ASR/projets/soutenances/Ranaivo-Sabourin/rapport-Simulation_P2P.pdf.

Ebel, H., Mielsch, L.-I. and Bornholdt, S. (2002), “Scale-free topology of e-mail networks”, *Phys. Rev. E* , Vol. 66, p. 035103.

Erdős, P. and Rényi, A. (1959), “On random graphs. I”, *Publ. Math. Debrecen* , Vol. 6, pp. 290–297.

Fishman, G. S. (1995), *Monte Carlo: concepts, algorithms, and applications*, Springer.

Fraiman, D. (2008), “Growing directed networks: stationary in-degree probability for arbitrary out-degree one”, *The European Physical Journal B - Condensed Matter and Complex Systems* , Vol. 61, pp. 377–388.

Gnutella (2001), *The Gnutella Protocol Specification v0.4*.

Goh, K.-I., Kahng, B. and Kim, D. (2001), “Universal behavior of load distribution in scale-free networks”, *Phys. Rev. Lett.* , Vol. 87, p. 278701.

Goh, K., Kahng, B. and Kim, D. (2005), “Evolution of the protein interaction network of budding yeast: Role of the protein family compatibility constraint”, *J. Korean Phys. Soc* , Vol. 46, pp. 551–555.

Grandison, T. and Sloman, M. (2000), “A survey of trust in internet applications.”, *IEEE Communications Surveys and Tutorials* , Vol. 3, pp. 2–16.

Granovetter, M. (1985), “Economic action and social structure: The problem of embeddedness”, *American Journal of Sociology* , Vol. 91, pp. 481–510.

Guardiola, X., Guimera, R., Arenas, A., Diaz-Guilera, A., Streib, D. and Amaral, L. A. N. (2002), “Macro- and micro-structure of trust networks”, *ArXiv Condensed Matter e-prints* .

Hoffman, K., Zage, D. and Nita-Rotaru, C. (2007), A survey of attack and defense techniques for reputation systems, Technical report, Purdue University.

Holloway, T., Bozicevic, M. and Börner, K. (2005), “Analyzing and visualizing the semantic coverage of wikipedia and its authors”, *CoRR* , Vol. abs/cs/0512085.

Husted, B. W. (1989), “Trust in business relations: Directions for empirical research”, *Business and Professional Ethics Journal* , Vol. 8, pp. 23–40.

Jesi, G. P. (2004), ‘Peersim: A peer-to-peer simulator’, http://peersim.sourceforge.net.

jimt (n.d.), ‘Efficiently selecting a random, weighted element’, http://www.perlmonks.org/?node_id=577433. last visited on 14/12/2008.

Jonczy, J., Wüthrich, M. and Haenni, R. (2006), A probabilistic trust model for GnuPG, *in* ‘23C3, 23rd Chaos Communication Congress’, Berlin, Germany, pp. 61–66.

Jøsang, A. (2001), “A logic for uncertain probabilities”, *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* , Vol. 9, pp. 279–311.

Jøsang, A., Gray, E. and Kinateder, M. (2006), “Simplification and analysis of transitive trust networks”, *Web Intelligence and Agent Systems* , Vol. 4, IOS Press, pp. 139–161.

Jøsang, A. and Pope, S. (2005), Semantic constraints for trust transitivity, *in* ‘Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling - Volume 43’, APCCM ’05, pp. 59–68.

Kamvar, S. D., Schlosser, M. T. and Garcia-Molina, H. (2003), The eigentrust algorithm for reputation management in P2P networks, *in* ‘WWW ’03: Proceedings of the 12th International Conference on World Wide Web’, ACM, pp. 640–651.

Kazaa (2006), http://www.kazaa.com. Last visited on 30/09/2010.

Klemm, K. and Eguíluz, V. M. (2002), “Highly clustered scale-free networks”, *Phys. Rev. E* , Vol. 65, p. 036123.

Krapivsky, P. L., Rodgers, G. J. and Redner, S. (2001), “Degree distributions of growing networks”, *Phys. Rev. Lett.* , Vol. 86, pp. 5401–5404.

Lehmann, S., Lautrup, B. and Jackson, A. D. (2003), "Citation networks in high energy physics", *Phys. Rev. E* , Vol. 68, American Physical Society.

Luhmann, N. (1988), "Familiarity, confidence, trust: Problems and alternatives", *D. Gambetta, editor, Trust: Making and Breaking of Cooperative Relations, Basil Blackwell, Oxford, 1988* .

Lui, S. M. and Kwok, S. H. (2002), "Interoperability of peer-to-peer file sharing protocols", *SIGecom Exch.* , Vol. 3, pp. 25–33.

Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S. (2002), Search and replication in unstructured peer-to-peer networks, *in* 'ICS '02: Proceedings of the 16th international conference on Supercomputing', ACM, pp. 84–95.

Mahoney, G., Myrvold, W. and Shoja, G. C. (2005), Generic reliability trust model, *in* 'Third Annual Conference on Privacy, Security and Trust'.

Mcknight, D. H. and Chervany, N. L. (1996), The Meanings of Trust, Technical report.

Michiardi, P. and Molva, R. (2002), Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks, *in* 'Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security', Kluwer, B. V., pp. 107–121.

Nandi, A., Ngan, T.-W., Singh, A., Druschel, P. and Wallach, D. S. (2005), Scrivener: Providing incentives in cooperative content distribution systems, *in* 'Middleware', pp. 270–291.

Ribeiro de Mello, E., van Moorsel, A. P. A. and da Silva Fraga, J. (2007), Evaluation of P2P search algorithms for discovering trust paths, *in* 'European Performance Engineering Workshop', Lecture Nodes in Computer Science, Springer, pp. 112–124.

Shen, X., Yu, H., Buford, J. and Akon, M. (2009), *Handbook of Peer-to-Peer Networking*, 1st edn, Springer Publishing Company, Incorporated.

Skype (2003), http://www.skype.com/intl/en-gb/home/. Last visited on 30/09/2010.

Srivatsa, M., Xiong, L. and Liu, L. (2005), Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks, *in* 'WWW '05: Proceeding of the 14th international conference on World Wide Web', ACM, pp. 422–431.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H. (2001), Chord: A scalable peer-to-peer lookup service for internet applications, *in* 'Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications', SIGCOMM '01, pp. 149–160.

Ulrich, A., Holz, R., Hauck, P. and Carle, G. (2011), Investigating the openpgp web of trust, *in* 'Proceedings of the 16th European conference on Research in computer security', ESORICS'11, pp. 489–507.

Vázquez, A., Flammini, A., Maritan, A. and Vespignani, A. (2003), "Modeling of protein interaction networks", *Complexus* , Vol. 1, pp. 38–44.

Vuze (2003), http://www.vuze.com/. Last visited on 30/09/2010.

Wallach, D. (2003), "A survey of peer-to-peer security issues", *Software SecurityTheories and Systems* , Springer, pp. 253–258.

Walsh, K. and Sirer, E. G. (2006), Experience with an object reputation system for peer-to-peer filesharing, *in* 'NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation', USENIX Association.

Williamson, O. E. (1993), "Calculativeness, Trust, and Economic Organization", *Journal of Law and Economics* , Vol. 36, The University of Chicago Press, pp. 453–486.

Xiong, L. and Liu, L. (2004), "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities", *IEEE Transactions on Knowledge and Data Engineering* , Vol. 16, pp. 843–857.

Yang, B. and Garcia-Molina, H. (2002), Improving search in peer-to-peer networks, *in* 'ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems', IEEE Computer Society, pp. 5–14.

Zhang, H. and van Moorsel, A. P. A. (2008), Evaluation of P2P algorithms for probabilistic trust inference in a web of trust, *in* 'Computer Performance Engineering: 5th European Performance Engineering Workshop', pp. 242–256.

Zhou, R. and Hwang, K. (2007), "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing", *IEEE Trans. Parallel Distrib. Syst.* , Vol. 18, pp. 460–473.

Zimmermann, P. R. (1995), *The official PGP user's guide*, MIT Press, Cambridge, MA, USA.