Topics in the structure and search of large networks

Siantos, Yiannis

Awarding institution:
King's College London

Topics in the structure and search of large networks

Yiannis Siantos
King's College London

Submitted for the degree Doctor of Philosophy
September 2014

Abstract

Through the development of the topic of Web Science there has been interest in the evolution of networks such as the WWW and online social networks almost as ecospheres in a biological sense. However much of the value of the web comes from our ability to search and index it rapidly through the development of ranking and retrieval algorithms such as that offered by Google. Our thesis examines properties of online social networks, and in particular Twitter, whose properties have not been examined to date. However one major problem is the very large size of these networks, and limited amount of resources that are usually available when accessing them for research purposes.

We show that through the use of random walks, we are able to quickly discover important portions of such networks and estimate interesting properties, while keeping computational costs low.

Our thesis focuses on the following:

1. To study how to crawl massive social networks representatively with limited resources. This will allow users to get a meaningful snapshot of the network. Our methodology for this is:

   (a) To investigate this problem experimentally and theoretically on artificial networks simulated in a controlled environment.

   (b) To investigate on real networks by comparing our limited designed crawls, with data we have obtained giving the complete structure of e.g. the Twitter network.

2. To investigate the graph theoretic structure of the networks, to the following ends:

   (a) To devise representative methods of generating artificial networks as given in 1(a), both experimentally, and supported by theory.

   (b) To relate this structure to improving the design of algorithms for information retrieval, ranking home pages, viral advertising etc.

3. To investigate the existence of patterns in user behavior in social networks in order to:

   (a) Detect user groups with similar interests/behaviors.

   (b) Recommend activities to users based on activities of similar users.

Acknowledgements

I firstly want to thank my supervisor Prof. Colin Cooper. His experience, knowledge and guidance throughout my Ph.D. have been invaluable. He has been an inspiration for my continuous desire to learn and improve both as a researcher as well as an academic.

I also wish to thank my second supervisor Prof. Tomasz Radzik. His continuous support and advice have been very important throughout my Ph.D.

I would also like to thank my colleague Mr. Sang Hyuk Lee. Collaborating with him has been very constructive and at the same time enjoyable experience.

In addition, I would like to thank by colleague Mr. Christos Hadjinikoli. His help and support as a colleague and friend have helped me throughout the entire Ph.D.

Furthermore I would like to thank my colleague Ms. Andrada Voinitchi who's support, advice and motivation helped me greatly throughout my research.

I would also like to thank all my colleagues, friends and relatives who have supported me throughout these past few years. Although I could not possibly mention all of them here, their support has been invaluable.

Finally, I would like to thank my parents. Their support throughout my entire life have made me into the person I am and it is due to them that I have been able to achieve everything I have achieved so far in my life.

Dedication

To my parents, to whom I owe everything I am and to my late sister, who's been a constant inspiration in my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ever since the prevalence of the Internet as a medium to exchange information between people situated in remote locations, starting from the router network all the way to the World Wide Web (WWW) and more recently Online Social Networks (OLSNs), many interesting structures have popped up. These networks serve different purposes, e.g. the router network redirects packets of information along the internet, while the WWW is a collection of webpages connected to each other using hyperlinks and OLSNs allow people to communicate with their friends and acquaintances and interact in a social context. While these structures appeared at different times to serve different purposes, when one models these networks as graphs then similarities can be observed and new shared properties are constantly being discovered.

The graphs of these networks share a major common characteristic which puts them in a special family of *scale-free graphs* with power-law degree distributions. Informally, these graphs show a structural invariance no matter how massive they are, that is, their size does not play a part in their local structure and global properties, which makes them scale-free. Additionally when viewed as graphs, the degrees of the vertices of these graphs (i.e. the number of connections/edges each vertex has to other vertices) are in most cases small, however there are very rare but important exceptions of vertices of very high degrees. More specifically, when one selects a vertex uniformly at random (*uar*) from such a network, then the probability distribution of the degree of this vertex

follows a long-tail or power-law distribution.

The motivation behind our work comes from the scale-free aspect of these networks as well as their very large size. We are interested in determining further network properties but at the same time we wish to do this quickly and efficiently. Therefore our specific interests in this topic are as follows:

1. Methods to obtain representative samples of massive networks with limited resources. This will allow us to get a meaningful snapshot of the network without the expense of an extensive sampling of hundreds of millions of home pages.
2. Investigating the graph theoretic properties and structure of scale-free networks
3. Generation of artificial graphs which fit the structure of the examined networks and analyze the generation processes.
4. Prediction of future states of these graphs by taking advantage of knowledge already acquired.

The contents of this thesis also appear in the following publications:

Colin Cooper, Tomasz Radzik, and Yiannis Siantos.

A fast algorithm to find all high degree vertices in graphs with a power law degree sequence. In WAW 2012 Proceedings, volume 7323 of LNCS, pages 165–178. Springer, 2012. Appears in Chapter 7 [37].

Colin Cooper, Tomasz Radzik, and Yiannis Siantos.

A fast algorithm to find all high degree vertices in power law graphs. In Proceedings of the 21st international conference companion on World Wide Web, WWW '12 Companion, pages 1007–1016, New York, NY, USA, 2012. ACM. Appears in Chapter 7 [36].

Colin Cooper, Tomasz Radzik, and Yiannis Siantos.

Estimating network parameters using random walks. In CASoN, pages 33–40. IEEE, 2012. Appears in Chapters 5 and 6 [35].

Colin Cooper, Tomasz Radzik, and Yiannis Siantos.

Fast low-cost estimation of network properties using random walks. In Algorithms and Models for the Web Graph, volume 8305 of LNCS, pages 130–143. Springer International Publishing, 2013. Appears in Chapters 5 and 6 [38].

Colin Cooper, Tomasz Radzik, and Yiannis Siantos.

Estimating network parameters using random walks. Social Network Analysis and Mining, 4(1), 2014. Appears in Chapters 5 and 6 [39].

Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos.

Recommender systems based on random walks. Technical report, Department of Informatics, King's College London, August 2013. Appears in Chapter 8 [33]. Part of the project "Fast low cost methods to learn structure of large networks", funded by Samsung as part of the Samsung Global Research Outreach programme awarded in 2012.

Colin Cooper, Sang-Hyuk Lee, Tomasz Radzik, and Yiannis Siantos.

Random walks in recommender systems: exact computation and simulations. In WWW (Companion Volume), pages 811–816. ACM, 2014. Appears in Chapter 8 [34].

1.1 Large Online Networks

The reason that OLSNs are interesting from a computer science and mathematical perspective may not be obvious. However upon deeper examination this issue is one which has motivated increasing interest over the past few years due to the explosive growth of OLSNs and the relative power that people have within these networks to affect the world around them.

Recent developments in technology have allowed the creation of large networks, which can be accessed globally via personal computers, or more recently mobile phones. The

original and most outstanding examples of such networks are the WWW, and the network of email exchanges. Very recently, many novel OLSNs such as Twitter and Facebook, or online video repositories such as YouTube have sprung up. These networks, extensively interleaved with each other and the WWW, have substantial impact on the way we live our lives. Nobody who has followed the political unrest in North Africa during February of 2011 can be unaware of the importance of e.g. Twitter in galvanizing and coordinating social behavior.

The WWW is remarkable in its own right, and its structure has been subject of considerable research to understand the phenomena which are observed such as, for example, the tendency of WWW pages to link to pages to which many other pages have already linked. More recently, through the development of the topic of Web Science there has been interest in the evolution of such networks as ecospheres in a biological sense. However much of the value of the web comes from our ability to search and index it rapidly through the development of ranking and retrieval algorithms such as that offered by Google.

Our research examines properties of OLSNs, to determine whether they share similar structural properties with the WWW or other social networks. We focus particularly on Twitter which is a very interesting OLSN from various aspects. The WWW is known to exhibit a scale-free structure, with small world properties such as small diameter. By understanding graph structure, smaller networks similar to the WWW can be modelled, to allow evaluation of search algorithms.

The results that we will be presenting from crawling these networks, combined with a presentation of the theoretical background show a similarity of basic structure between the WWW and these other social networks with respect to properties such as degree frequency distribution etc. This similarity may arise because of the way these networks were generated. The similarity of the structure may allow the creation of a generative model which describes them, and exhibits the same characteristics.

One major problem is the very large size of these networks, and limited amount of

resources that are usually available when accessing them for research purposes. To solve this, a way to take representative samples will need to be found which ideally will produce samples relatively small, to give a correct indication of specific properties of interest.

The areas of online social graph structure, unknown graph sampling and graph generation, mentioned above are unresolved topics, and many open questions remain. Solving these problems by developing good theoretical graph generation methods and proposing an effective and efficient way to sample online graphs, are of interest and would be a useful tool for the community.

1.2 Searching and sampling large online networks

Another important aspect of our research focuses around graph sampling and crawling. In reality we do not differentiate between these two terms since they are often used to express the same goal. The remarkable size to which real world networks have grown means that it is no longer feasible to obtain entire networks. Even if it were feasible however, it may not be desirable due to the fact that in an ideal world we would like to minimize the "cost" of obtaining these networks and most of the time this means obtaining a precise and small sample.

Graph sampling and crawling can be seen as two terms which are similar in many ways. Both terms are usually used in the same context and while sampling is the goal, crawling is the way to achieve it. There are numerous methods available to gather samples from graphs and many of them can be applicable to the real world networks. These methods include Random Walks (RWs) and their variations as well as Breadth-First Search (BFS) and other similar methods. Many of these methods will be presented throughout this thesis as they play an important role in achieving some of our fundamental goals which is graph sampling.

In our work we make extensive use of Markov Chains on graphs (Random Walks) and related methods. We rely on simulating random walks on manufactured graphs or real

online networks. The advantage of using RWs for the purposes of sampling from large online network are:

- The RW naturally fits the constraints typically imposed by these networks e.g. most networks don't allow random access to their users but allow the neighboring users of a given user to be queried.
- Markov Chains in general are memoryless and therefore we do not need to store too much information locally. This is a great advantage since we use a personal computer with limited capabilities which would, in practice, not be able to store partial BFS trees or similarly large structures.
- There is a rich assortment of resources regarding RWs, such as e.g. the book by Aldous and Fill [2], the survey by Lovász [95] or the work by C. Cooper *et al.* [30, 32].

We use RWs for various reasons: (a) sample from a network with the purpose to estimate certain properties, (b) quickly discover important vertices in a network and (c) rank all other vertices in a network according to their relevance to a given vertex.

1.3 Network Modeling and Generation

While the intuition for modeling an OLSN as a graph is rather simple and may even seem trivial, one may discover that upon interpreting such a structure there are many questions that automatically arise. Such questions may include:

- What are the underlying processes that take place and contribute to the visible structure of the network?
- Why does the network have this particular rate of growth?
- How do individual actors interact within the network and how does this affect the network structure?

- What is the most likely structure of the network after some time has elapsed?
- Who is most likely to create more links on the network and who is more likely to receive those links?
- How can we get a sample and how do the properties of this sample relate to the properties of the entire network?

These are just a few of the questions that arise. In general modeling the real world networks is still an open question. There are numerous models which are available to generate graphs which match a wide range of properties of real world networks (such as OLSNs) but there is still work to be done with new observations constantly being made.

Generative models work based on various intuitions and can be seen as network evolution models in the sense that they initialize small graphs and then build upon them in the same way as a real network starts with a single node and evolves over time. It is believed that this aspect of growth is essential in modeling these sorts of networks but it is not the only aspect which needs to exist in these models.

1.4 Contributions and thesis outline

In this section we will outline our contributions to the areas of graph generation, RW simulation, sampling, network search and information filtering.

1.4.1 Graphs properties and generation

In Chapter 2 we investigate graph generation models which generate graphs with properties that resemble real large online networks. We review various existing models and introduce the following new ones:

1. Random walk graph model (Section 2.2.8).
2. Preferential attachment with message propagation model (Section 2.2.9).

3. Grow, back-connect, densify model (Section 2.2.10).

4. Partitioned preferential attachment model (Section 2.2.11).

5. Implicit Power-Law graph model (Section 2.2.12).

1.4.2 Markov Chains and Random Walks

As mentioned above, Markov Chains are used extensively in our work. In Chapter 3 we introduce some basic properties of Markov Chains and in addition, some special cases of Markov Chains i.e. the Simple Random Walk (SRW), Weighted Random Walk (WRW), Lazy Random Walk (LRW) and Metropolis Hastings Random Walk (MHRW). Our main contribution in the area of Markov Chains is in Markov Chain simulations. In Section 3.6 we discuss the algorithmic aspect of RW simulations. We also propose a method to greatly speed up WRWs when ran on large online networks using appropriate caching techniques.

1.4.3 Estimating network properties

Estimating network properties is an area which is of great interest. As mentioned above, due to the large size of large online networks, it is generally impractical, and sometimes impossible to obtain entire snapshots of them, using the limited resources we have. At the same time we are interested in knowing some of their basic properties. By this point, it is clear that a solution to this problem needs to be found, and in our case, the solution is sampling these networks, using appropriately designed sampling methods which, by their design, would allow us to infer the properties we are interested in.

A general outline of some existing methods to estimate network properties can be found in Chapter 4. Specifically, in this chapter we discuss some methods originating from the fields of zoology, anthropology and sociology. In addition we present work we have done in sampling from large online networks, and specifically Twitter [125]. In Section 4.3, we show some of the more subtle differences of sampling using a RW against sampling

uar. We also show certain drawbacks of uniform sampling by showing the rate of obtaining certain categories of vertices on synthetic graphs which are similar to large online networks.

In Chapter 5 we present additional methods to estimate network properties. These methods differ from the methods presented in Chapter 4 which are, in most part, methods based on sampling *uar*. The methods presented in Chapter 5 are sampling methods based on RWs. We present an in depth description of three methods:

- The method of the first returns.
- The method of the Cycle formula of regenerative processes (CFRP).
- The method of the running totals.

In Chapter 6 we present experimental results obtained when applying these methods on manufactured graphs as well as large online network datasets.

1.4.4 Network search and discovery

In Chapter 7 we present a method we have devised to quickly discover all high degree vertices of a web-graph quickly. We suggest a method based on a degree biased RW which would discover such vertices quicker than a SRW. We show both theoretically and experimentally that this method indeed discovers high degree vertices quicker than a SRW on graph generated using the web-graph model. In addition, the cover time of the entire graph does not increase compared to a SRW. We also show that this method works well as a heuristic to discover high degree vertices quickly on large online networks.

1.4.5 Information Filtering and Recommendation

An information filtering system, is a system which, when provided with data, will filter the data, keeping only the relevant information contained within, while discarding irrelevant or redundant information. These systems have many applications such as in

signal processing, information extraction and recommender systems. In Chapter 8 we present our work in information filtering, and specifically in recommender systems. In particular we focus on recommender systems based on collaborative filtering and build on the work by Fouss *et al.* [50, 51].

Our contributions to this area are:

- We show that the a simple approach based on the transition probability matrix of a RW outperforms many of the state-of-the-art recommendation algorithms.
- We suggest ways to optimize the quality of the recommendations which are produced based on on the transition probability matrix of a RW.
- We show how using different types of WRW further improves the quality of the recommendations generated.
- We further improve on the space efficiency of existing methods by simulating RWs instead of using exact matrix operations. This results in estimated results which rapidly converge to the expected value, require less physical memory, and are obtained without adding any additional time complexity to the algorithms.

Chapter 2

Scale-free graphs and models

In this chapter we mainly focus on a special family of graphs known as *scale-free graphs*. These graphs have emerged as models in the area of large online networks, such as the WWW and OLSNs. They are also of interest in many areas of research, such as the social sciences, mathematics, physics, biology, bioinformatics and computer science. We mention the area of physics and biology as being interested in these graph structures because they tend to be similar to to other structures observed in the physical world, such as protein interaction networks as well as particle interaction networks.

This particular area of research is relatively new in Computer Science (CS). This is mainly due to the fact that it is only relatively recently that scale-free network structures have emerged as important in CS. It started with the emergence of the WWW, however work on graph generation models greatly pre-dates the rise of the WWW, and was previously used to model social networks in the domain of social sciences.

The WWW graph in particular has received a great deal of attention. This graph is the result of modeling the WWW as a set of pages which are the vertices of the graph, and a set of directed links between the pages which are the edges of the graph [72]. Some very interesting properties that have been discovered in this graph, for example the degree distribution observed follows a long-tail distribution. This is apparently a common attribute of large networks that is present in the WWW graph [4, 16, 72, 75], as well as the Internet (autonomous systems) [47], citation graphs [118], OLSNs [47] and

many others.

This Chapter is in 2 sections. The first section, Section 2.1 (Graph properties of large networks), outlines various graph properties which are considered to be important or representative of large networks. The second section, Section 2.2 (Scale-free graphs and related models), reviews some typical models currently regarded as interesting. This section also details simulations and experiments we made in this area. These experiments were done mainly on a system having an Intel Core 2 Quad and 8GB memory, which are capabilities available on most personal computers today. Where necessary, we have also made use of a workstation equipped with 2 Intel Xeon 12 Core processors (24 processors in total) and 64GB of memory.

2.1 Graph properties of large networks

We next outline various properties of these graphs which are perceived as being of interest.

The term of "small world" networks is widely used in popular culture to reflect large graphs with low edge density but good connectivity with clustering properties and where average distances are small.

2.1.1 Degree Sequence

A graph with n vertices can be characterized by its degree sequence $D_0, D_1, \ldots, D_{\Delta(G)}$ where D_j is the number of vertices of degree j in the graph and $\Delta(G)$ is the maximum degree of the graph.

2.1.2 Degree distribution

We view the degree sequence as a probability distribution where we assume a stochastic experiment of sampling a graph vertex u, *uar*. The degree distribution defines the *probability that vertex u has degree X*. In OLSNs and the WWW it has been observed

that this probability follows a negative exponential power-law, specifically:

$$P(d(u) = X) = \frac{A}{X^{\gamma}} \quad \gamma > 1 \tag{2.1}$$

where $d(u)$ is the degree of vertex u and A is a positive constant.

In Equation (2.1) the constant γ is often called the *parameter of the power-law* and while all observed OLSNs, the WWW as well as the internet topology network [47], exhibit aspects of a power-law degree distribution, they often have different parameters for γ. Additionally, depending on whether the graph is directed or undirected, there can be different values of the constant γ for the in and out-degree. For example for the world-wide web the observed in-link distribution follows a power-law with a γ ranging from 2 to 2.5 while the out-link distribution ranges from 2.3 to 3 [4, 16, 72, 75]. As an empirical rule of thumb it has been observed that in OLSNs the value of γ ranges between 1.5 and 3 while for the WWW it has been found to be approximately 3, when the network is modeled as an undirected graph.

2.1.3 Diameter

The diameter of the aforementioned networks seems to be relatively small and this is taken as evidence that a "small world" phenomenon has been observed. This is usually taken to mean that the diameter d, or effective diameter [119] is proportional to $\log n$ i.e.

$$d \propto \log n \tag{2.2}$$

where n is the size of the graph.

However according to a study by J. Leskovec *et al.* [68] it was proposed that the diameter is even smaller than this proposed value. It is unclear whether or not this observed characteristic of certain networks is related to another observed characteristic of an edge densification power-law [68] where $|E(t)| \propto |V(t)|^{\alpha}$. Here $E(t)$ and $V(t)$ are the sets of edges and vertices at time t respectively, and α is non-trivial ($\alpha > 1$). The claim however is that the diameter is upper bounded by $d = \Theta(\log n)$.

2.1.4 Scale-Free Structure

Having introduced the concepts of degree distribution and the small world diameter, we now clarify what we mean by *scale-free graph*. The graphs having properties of a small world diameter (2.2) and a power-law degree distribution (2.1) are the types of graphs we will refer to as *power-law* graphs or *scale-free* graphs. The latter term is controversial however so for simplicity's sake we will use the term power-law graphs and scale-free graphs interchangeably to refer to all graphs which have a power-law degree distribution, and a diameter which is bounded by the expected diameter of a small-world network, i.e. Diam$G = O(\log n)$. Additionally we require graphs which we refer to as scale-free to have a scale invariance in the size of the graph, where these basic properties remain true even at very different sizes of the graph. According to the work of Dill *et al.* [41] the web graph has a self-similar structure which may be the cause of the aforementioned scale invariance of the web-graph. Having a self-similar structure means that, if we partition the graph in sub-graphs, then these sub-graphs will be similar to the entire graph with respect to (w.r.t.) their properties. It may be reasonable to assume that many other scale-free graphs observed such as OLSN graphs share this characteristic with the web-graph.

2.1.5 Degree correlations

The term "degree correlation", sometimes referred to as neighbour assortativity, is used to denote the relationship between the degree of a vertex and the degree of its neighbours. In the work of Pastor-Satorras *et al.* [114] it has been suggested that the degree correlations in real world networks, such as the WWW, are different from what would be expected in random graphs. The metric defined in [114] was based on the conditional probability $P_c(k' \mid k)$ which denotes the conditional probability that a vertex u of degree $d(u) = k$ is connected to a vertex v with degree $d(v) = k'$. As stated by Pastor, due to statistical fluctuations, measuring this probability is a rather complex task. He

suggested the following alternative:

$$\langle k_{nn} \rangle = \sum_{k'} k' P_c(k' \mid k) \tag{2.3}$$

The value of $\langle k_{nn} \rangle$ denotes the average degree of the vertices to which vertices of degree k are connected. What has been noted in [114] is that the relationship between k and $\langle k_{nn} \rangle$ follows a power-law, i.e. is of the form $\langle k_{nn} \rangle \propto \frac{1}{k^c}$. This dependence has been observed in various snapshots of the WWW graph, where the power-law constant was measured to be $c = 0.5$.

2.1.6 Expander graphs

The expansion property of a graph is a property we will be using frequently. In general when we talk about the expansion property of a graph, or if we describe a graph as an *expander*, this may indicate one of several different properties which may hold for a graph. Expander graphs were first defined by Bassalygo and Pinsker [7] in the early 70s.

There are several expansion properties that we can take into account:

Edge Expansion. The edge expansion $h(G)$ is sometimes referred to as the isoperimetric number or Cheeger constant; a metric which corresponds to graph conductance, described in more detail in Section 2.1.7. We note that the notion of edge expansion is the same as conductance which will be further discussed in Section 3.1.3.

The edge expansion of a set of vertices S belonging to a graph G given by:

$$h(G) = \phi_G = \min_{\substack{S \subset V \\ |S| \leq \frac{|V|}{2}}} \phi(S) \tag{2.4}$$

where $\phi(S)$ is the conductance of set S given by:

$$\phi(S) = \frac{\sum_{i \in S, j \notin S} a_{ij}}{\min(\alpha(S), \alpha(\overline{S}))} \tag{2.5}$$

$$\alpha(S) = \sum_{i \in S} \sum_{j \in V} a_{ij} \tag{2.6}$$

Where a_{ij} is the entry (i, j) in the adjacency matrix of the graph and $\alpha(S), \alpha(\overline{S})$ are the degree of set S and $V \setminus S$ respectively.

According to J. Dodziuk [43] there is a direct correspondence to the expansion properties of a d-regular graph with the eigenvalues of its adjacency matrix given by the inequality:

$$\frac{d - \lambda_2}{2} \leq h(G) \leq \sqrt{2d(d - \lambda_2)} \tag{2.7}$$

where $d - \lambda_2$ is the gap between the largest and second largest eigenvalue of the adjacency matrix.

Vertex Expansion. The vertex isoperimetric numbers or vertex expansions $h_{\text{out}}(G)$ and $h_{\text{in}}(G)$ are defined as follows:

$$h_{\text{out}}(G) = \min_{0<|S|\leq \frac{n}{2}} \frac{|\partial_{\text{out}}(S)|}{|S|} \tag{2.8}$$

$$h_{\text{in}}(G) = \min_{0<|S|\leq \frac{n}{2}} \frac{|\partial_{\text{in}}(S)|}{|S|} \tag{2.9}$$

where $\partial_{\text{out}}(S)$ denotes the *outer boundary* of S i.e. the set of vertices $\{u : u \in V \backslash S, \exists e = \{u, v\}, v \in S\}$. Conversely $\partial_{\text{in}}(S)$ denotes the *inner boundary* of S i.e. the set of vertices $\{u : u \in S, \exists e = \{u, v\} v \in V \backslash S\}$.

In general when we talk about expander graphs we refer to the edge expansion $h(G)$ and we call a graph with $h(G) = \alpha$ an α-*expander*.

2.1.7 Connected Components And Community Structure

Recent work by Girvan *et al.* [56] has also suggested the existence of distinct community structure within OLSNs (e.g. the collaboration network) the WWW as well as biological networks (specifically the "food web of marine organisms living in the Chesapeake Bay" [56]). When we talk about communities it is useful to describe what we are referring to. Generally there are many varying definitions in graph-theoretical terms of what a community is and how we detect it.

The first and simplest method used is finding connected components within the graph using algorithms such as breadth-first search or other more optimized algorithms for this purpose. Connected components in a directed graph is ambiguous and therefore we clarify by using the terms Weakly Connected Component (WCC) and Strongly Connected Component (SCC). A SCC is a component (sub-graph) $G_s = \{V_s, E_s\}$ of a given graph G where for each vertex pair $u, v \in V_s, u \neq v$ there exists a directed path connecting them. The related concept of WCCs differs in that vertices u, v can be connected by either a directed or undirected path. In the case of directed graphs, an undirected path is a path that is obtained by ignoring the edge direction. Since the notion of "community" mainly stems from the social sciences, the idea of partitioning graphs into components does not provide with a partitioning which is informative in the context of community detection in OLSNs. In such networks, and in the context of the social sciences, the term community is used to indicate sub-graphs which are internally densely connected while externally sparsely connected.

Communities: Conductance

In the context of OLSNs, communities can be very different from simply the SCCs or WCC, of the graph. For example consider a group of vertices S in a graph which have a degree sum of $\sum_{v_i \in S} d(u_i) = |E(S)|$ where $d(u_i)$ is the degree of vertex u_i. If the majority of the edges that are contained in S are edges which connect vertices within S then we can reasonably consider this group of vertices S to be a "good" community. This is also mentioned and analysed in the work of Leskovec *et al.* [89] where the *conductance* ϕ measure is used as the quantitative measure of the quality of the community. This gives a numerical value to the comparison mentioned above. Using this measure we can give meaning to the concept of comparing community quality, by comparing their conductance ϕ. We remind that conductance, first introduced in Section 2.1.6, is given by $\phi(S) = \frac{\sum_{i \in S, j \notin S} a_{ij}}{\min(\alpha(S), \alpha(\overline{S}))}$ where $S \subseteq V$ and $|S| \leq \frac{|V|}{2}$.

In the context of Markov Chains, the conductance of a set S as well as of a graph G is a quite interesting property. We explain the importance in more detail in Chapter 3 and

specifically in Equation (3.15), where we discuss how conductance can be used to bound the *mixing rate* of a RW which is a very important property for our investigation.

Communities: Centrality and modularity

As suggested in the work of M. Girvan *et al.* [56] we can also measure community structures using alternative network measures such as betweenness centrality, closeness centrality etc. The betweenness centrality C_B is a measurement of how many optimal paths go through a specific vertex, more formally:

$$C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(u)}{\sigma_{st}} \tag{2.10}$$

where σ_{st} is the number of shortest paths from s to t, and $\sigma_{st}(u)$ is the number of shortest paths from s to t that pass through a vertex u.

The idea is that we partition the graph by removing edges with high betweenness centrality we will separate the graph into a number of disjoint sets of vertices, each set being a good community.

There is also a measure suggested by J. Newman *et al.* [112] called *modularity* which gives a quantitative measurement of the quality of a given partitioning of a graph. This measure effectively compares the number of edges which are present within a given partition of the graph with the expected number of edges that would be found in a random graph with the same number of vertices and edges.

Assuming a graph partitioned into c non-overlapping sets, the textbook measurement of modularity as defined in [112] is shown below:

$$Q = \sum_{i=1}^{c} (e_{ii} - a_i^2) \tag{2.11}$$
$$a_i = \sum_{j=1}^{c} e_{ij}$$

where e_{ii} is the *fraction* of edges within the i-th set of the partition and a_i corresponds to the fraction of endpoints of edges attached to vertices in i.

As stated in [112], in a network where the number of within-community edges is approximately the same as the number of within-community edges in a random graph with the same vertices and community split then the modularity is 0 while values approaching 1.0 indicate a very strong community structure. In practice values over 0.3 are considered to denote a community structure.

The problem with the modularity measurement or partitioning in general, is that finding the optimal separation is an NP-complete problem. Recent work on approximation techniques [11] has made the estimation of near optimal communities feasible on very large graphs. This was achieved by attempting to iteratively optimize the modularity of a given partition by moving vertices into other communities, effectively producing near optimal partitioning in polynomial time.

Communities: Overlapping communities

In related work by N. Mishra *et al.* [106] it was suggested that the above measures are not sufficient to effectively represent the community structure which is apparent in OLSNs. In their work it was suggested that communities need not be disjoint and may, in fact, be overlapping. This is intuitively reasonable due to the fact that an average user of an OLSN network has an array of interests, each one belonging to a different category (e.g. sports, politics, region, religion) and in fact OLSN users belong to multiple communities based on their interests. They introduce the concept of (α, β)-*communities* which are formally defined as follows:

Definition 2.1. *We denote with $E(u, S)$ the set of edges with one endpoint on u and the other endpoint within set S. Given a graph $G = \{V, E\}$ in which every vertex has a self-loop. $C \subset V$ is an (α, β)-*cluster *if it is:*

1. Internally dense: $\forall u \in C, |E(u, C)| \geq \beta|C|$;
2. Externally sparse: $\forall u \in V \setminus C, |E(u, C)| \leq \alpha|C|$.

Additionally given $0 \leq \alpha < \beta \leq 1$ the (α, β)-*clustering problem* is defined as the problem to find all (α, β)-*clusters*.

This concept led to a generalized way of describing communities and analysing OLSNs. An example of this is in the work of J. He *et al.* [64] who suggested a method to detect such clusters. The idea is based on iteratively optimizing a randomly selected set of vertices S, by swapping vertices within S with vertices outside S until the conditions of β internal density and α external sparsity are reached. It has been shown in [64] that such a community structure is present in some OLSNs such as Twitter, SlashDot and the citation graph. In addition it is shown that (α, β)-communities are not generally observed in generated random graphs such as the preferential attachment graph (seen in Section 2.2.2).

2.1.8 Transitivity Ratio and Clustering Coefficient

The transitivity ratio and clustering coefficient are closely related a measures which are used to indicate how tightly vertices cluster together by giving an indicative measure based on cycles of length 3 (triangles).

First, let t_G be the number of triangles in G, and W_G the number of paths of length 3 (either cyclic and acyclic). The transitivity ratio is given by:

$$C = \frac{3t_G}{W_G} \tag{2.12}$$

The local clustering coefficient was first defined by Watts and Strogatz [128]. Assume a vertex u has $|N(u)|$ neighbours. The maximum number of possible edges between those neighbours are $\frac{|N(u)|(|N(u)|-1)}{2}$. The local clustering coefficient is the ratio of actual connected neighbours of u over the maximum possible connections. Specifically, if the graph is simple:

$$C_u = \begin{cases} \frac{2|\{e_{vw}\}|}{d(u)(d(u)-1)} & \text{if } d(u) > 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

where $v, w \in N(u)$ and $\{e_{vw}\}$ the set of edges among neighbours of u.

By using Equation (2.13) the average local clustering coefficient is defined as:

$$\frac{1}{n} \sum_{u \in V} C_u.$$

In practice, the clustering coefficient has no meaning for non-simple graphs since if we allow multiedges, the maximum number of connections between vertices in $N(u)$ are infinite. Therefore we only consider C_u on simple graphs.

2.2 Scale-free graphs and related models

We review various generative models, and identify their relevant graph properties or, more often, the shortcomings of the models. The discussion here includes experimental investigation. We remind that all experiments were mainly done on an Intel Core 2 Quad and 8GB memory, which are capabilities available on most personal computers today. Where necessary, we have also made use of a workstation equipped with 2 Intel Xeon 12 Core processors (24 processors in total) and 64GB of memory.

In addition we note that the graph generation process was coded in C#.NET [105]. T he degree distribution for each generated graph was acquired in MATLAB [101] and plotted using Gnuplot [129]. In all cases, wherever a seed graph is required as the base of a generation model, we use a complete graph of m vertices (unless otherwise stated), where m is the number of edges which are added at each step of the generation model (if applicable). In addition, graphs which we considered interesting, were visualised using Gephi [54] and the layout was done using the ForceAtlas 2 Algorithm (included by the creators of Gephi). This is a force-directed graph drawing algorithm which simulates a physical system based on the graph. In this system, the edges act as an attractive force between vertices, while the vertices themselves have a repulsive force between them. The graph drawing algorithm then iteratively moves each according to the forces applied to it, until the system reaches a balance.

2.2.1 Erdős–Rényi Graph

In graph theory, the Erdős-Rényi model consists of two models for generating random graphs. It is perhaps the simplest of all graph generation models and it can be seen

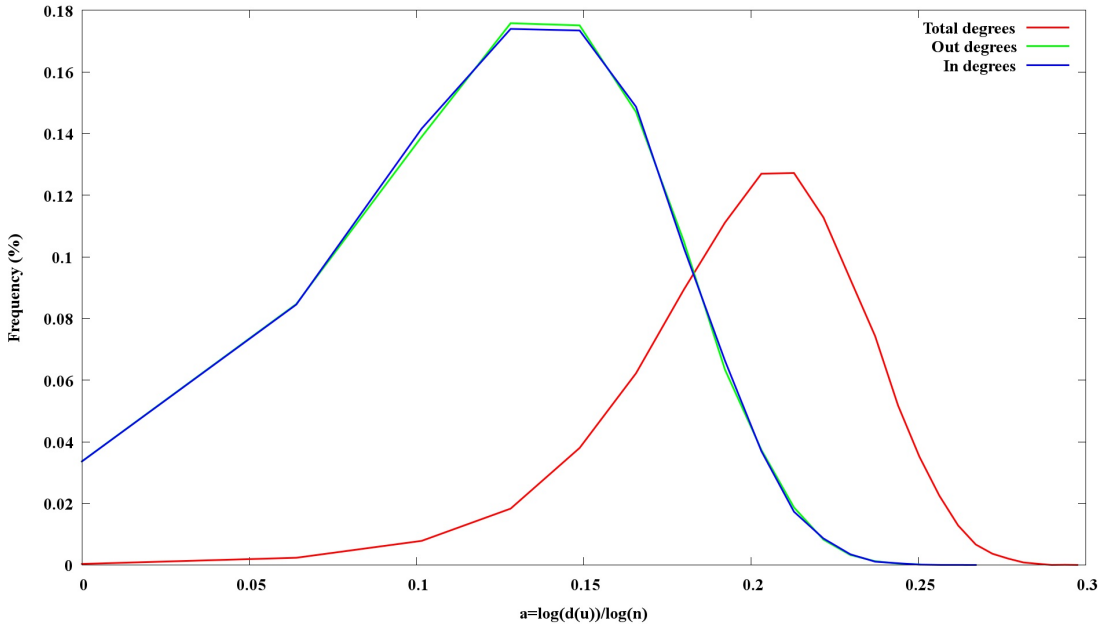


Figure 2.1: Erdős-Rényi Graph $G(n,p)$ graph with $n = 50000, p = 0.0001$

from two different aspects, the first one called the $G(n,M)$ model and the second one the $G(n,p)$ model. These models were examined by P. Erdős *et al.* [45].

$G(n,M)$ **model** From all the possible graphs with n vertices and M edges we choose one *uar*.

$G(n,p)$ **model** Consider a graph with n vertices. For each vertex pair u,v there exists an edge (u,v) with probability p

These models are very well analysed and understood. They tend to be connected when still remaining reasonably sparse ($p > \frac{\log n}{n}$). However they lack other basic properties that we need to successfully model an OLSN graph. For example, on average, they do not have a power-law degree distribution [45].

An example degree distribution of this graph can be seen in Figure 2.1.

2.2.2 Preferential attachment models

In this section we will describe a set of graph generation models referred to as *preferential attachment models*. These models are interesting in our work, since they are the basic models proposed to generate scale-free graphs, i.e. graphs which have a power-law degree distribution and small diameter regardless of the size of graph generated.

The term *preferential attachment* is used to describe the way that the edges are added in these graph generation models. Assume that at some point in one such model we wish to add an edge with one endpoint adjacent to vertex u. Preferentially selecting the other endpoint of the edge, means the probability $p_u(v)$ that edge (u, v) is added is equal to:

$$p_u(v) = \frac{d(v)}{2|E|} \tag{2.14}$$

which means v is selected proportionally to its degree.

An additional element required for a preferential attachment model to generate scale-free graphs is *growth* i.e. the graph size is not static but rather increases over time. Further details will be discussed when the *Barabási-Albert graph generation model* model and the generalized web-graph models are described.

Surveys by Bollobás and Riordan [12] and Drinea, Enachescu and Mitzenmacher [44] give many related generative procedures to obtain graphs with power-law degree sequences.

Barabási-Albert graph generation model and generalized Web-Graph Model

The *Barabási-Albert graph generation model* (BA model) is a graph process which generates graphs with degree distributions which follow a power-law. This generative method was proposed by Barabási and Albert [4] as a generative procedure for a model of the WWW.

It was empirically shown by Barabási that there are two required elements in order in order to obtain a power-law degree distribution. Two models were defined as follows:

Model A This model retains growth but all new vertices created are attached *uar* on existing edges.

Model B This model consists of a graph with a static vertex count which at each step an edge is added *preferentially*.

Each of the individual models above fails to create scale-free graphs, and it is shown in [4] that only by a combination of models A and B do we obtain a power-law degree distribution. The combination of these two models is what we refer to as a preferential attachment model. The generated graph is denoted by be $G(m,t)$ meaning the graph generated after t steps with m edges added at each step. Often we will let m be implicit and simply write $G(t)$. Adding an edge preferentially means that one or both end points of that edge is chosen preferentially i.e. according to Equation (2.14). Specifically for the *Barabási-Albert graph generation model* (BA model), assuming that we are adding m new edges at each step then the graph generated at step $t-1$ would have $2m(t-1)$ edges. Thus, at step t the probability $p(v,t)$ that vertex $v \in G(t-1)$ is chosen as an end point of a given edge is equal to:

$$p(v,t) = \frac{d(v,t-1)}{2m(t-1)}$$

Let $m \in \mathbb{N}$ and we have a graph $G(m,t-1)$ which is the graph generated after $t-1$ applications of the Barabási-Albert graph generation procedure, we can obtain graph $G(m,t)$ by doing the following:

- Select m vertices $\{v^{(1)}, v^{(2)}, \ldots, v^{(m)}\}$ from $G(m,t-1)$ preferentially, meaning that $v^{(i)}$ is chosen according to Equation (2.14).
- Add vertex v_t to $G(m,t-1)$
- Add edges $\{\{v_t, v^{(1)}\}, \ldots, \{v_t, v^{(m)}\}\}$.

A graph generated in this way has a power law of 3 for the degree sequence, irrespective of the number of edges $m \geq 1$ added at each step. The preferential edge endpoint

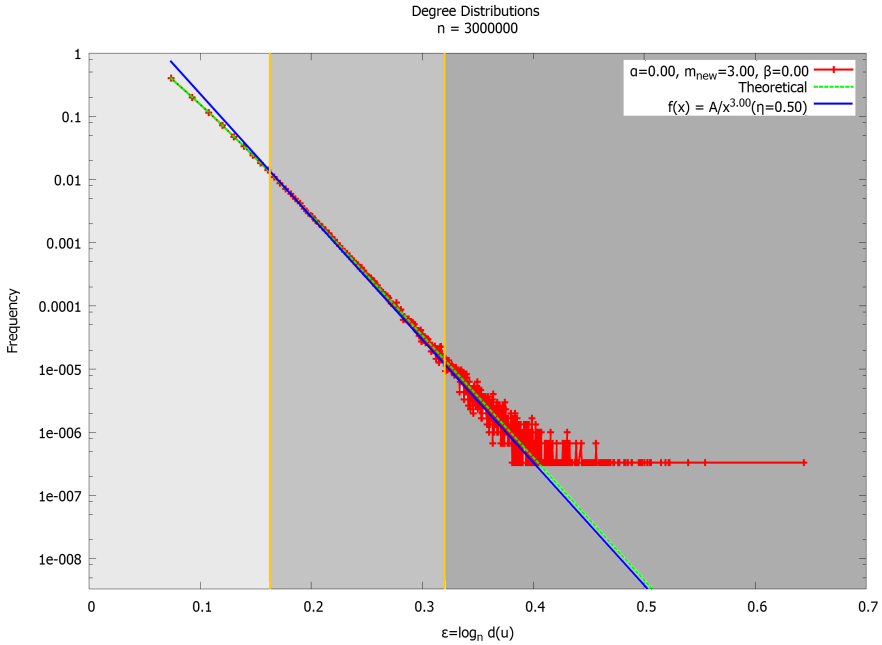


Figure 2.2: Degree distribution of a BA model graph

selection, and growth, are the reasons why the Barabási-Albert model is classified as a *preferential attachment model*.

We call graphs generated using this model, as well as the generalized web-graph model (see Section 2.2.2) *preferential attachment graphs*. Preferential attachment graphs have a heavy tailed degree sequence. Thus, although the majority of the vertices have constant degree, a very distinct minority have very large degrees. This particular property is the significant defining features of such graphs. A log-log plot of the degree sequence breaks naturally into three parts. This is illustrated in Figure 2.2. In this Figure we can see the lower range (small constant degree) where there is a slight curvature, as the power law approximation is incorrect. The middle range, of large but well represented vertex degrees, which give the characteristic straight line log-log plot of the power law coefficient. In the upper tail, where the sequence is far from concentrated, and the plot is a spiky mess. Due to the simple structure of the model as well as the extensive analysis that has been done, this will be the main reference model for our experiments.

General Undirected Web-Graph Model

The Barabási-Albert model was refined Bollobas and Riordan [13,14] who introduced the scale-free model and made detailed calculations of degree sequence and diameter. The model was generalized by many authors, including the web-graph model of Cooper and Frieze [24]. The web-graph model is very general and allows the number of edges added at each step to vary, for edges from new vertices to choose their end points preferentially or uniformly at random, as well as for insertion of edges between existing vertices. By varying these parameters, preferential attachment graphs with degree sequences exhibiting power laws c in the interval $(2, \infty)$ are obtained. Further details about the generalized web-graph model can be found in the next Section.

The general undirected web-graph model described in [24] can be seen as a general preferential attachment model. This model requires the parameters: $\alpha, \beta, \gamma, \delta, \boldsymbol{p}, \boldsymbol{q}$, where $\alpha, \beta, \gamma, \delta \in [0 \dots 1]$ and $\boldsymbol{p}, \boldsymbol{q}$ are probability distributions with $\boldsymbol{p} = (p_i : i > 1)$, $\boldsymbol{q} = (q_i : i > 1)$, $p_i, q_i \in [0 \dots 1]$ and $\sum_{i=1}^{\infty} p_i = 1$ and $\sum_{i=1}^{\infty} q_i = 1$. Considering the following events X_i= "There are i edges added", then $P_{NEW}(X_i) = p_i$ and $P_{OLD}(X_i) = q_i$.

We make use of the notation $G(t)$ as a shorthand notation to $G(\alpha, \beta, \gamma, \delta, \boldsymbol{p}, \boldsymbol{q}, t)$ to indicate the graph which is generated up to step t using the general web-graph process. As we mentioned above this graph is obtained by applying one step of the generation procedure on graph $G(t-1)$. In this case each step of the generative procedure is as follows:

- With probability α an existing vertex generates edges (Procedure OLD)
- With probability $1 - \alpha$ a new vertex is added and generates edges (Procedure NEW)
- Procedure NEW
    1. With probability p_i following $\boldsymbol{p}$ we will select i vertices from $G(t-1)$. The selection policy of these vertices is as follows:

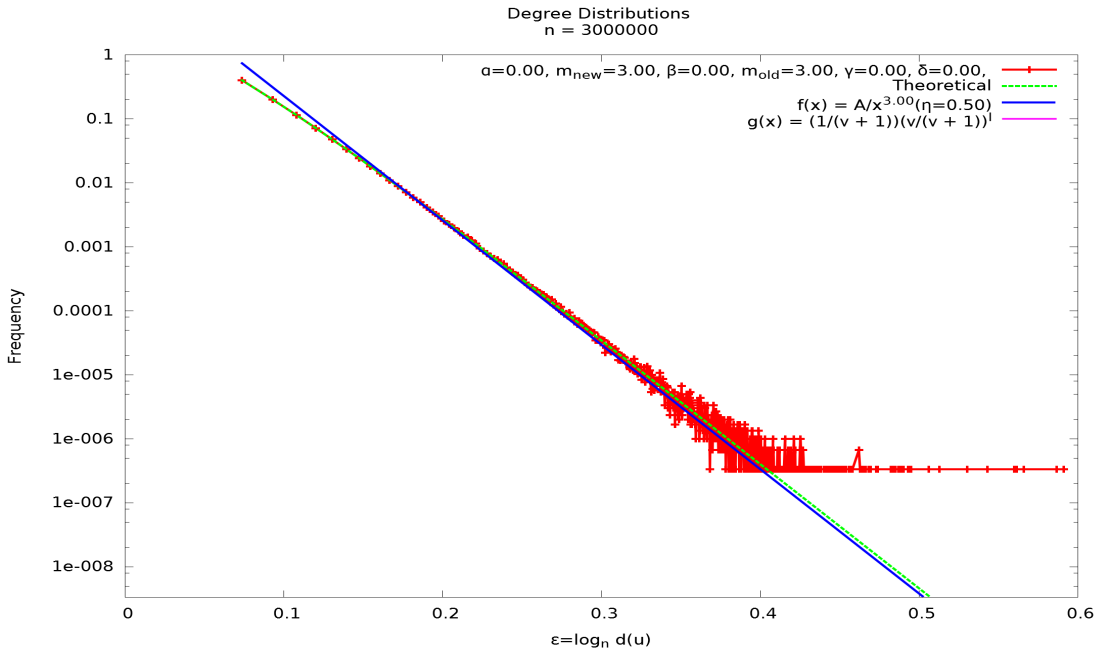


Figure 2.3: Degree distribution for $m = 3, c = 3.0$ (Barabási-Albert model)

With probability β for each of the vertices $\{v^{(1)}, v^{(2)}, \ldots, v^{(i)}\}$ that vertex is chosen *uar* and with probability $1 - \beta$ the choice is preferential.

2. Add vertex v_t to $G(t-1)$
3. Add edges $\{\{v_t, v^{(1)}\}, \ldots, \{v_t, v^{(i)}\}\}$.

- Procedure OLD

1. With probability δ we select a vertex v from $G(t-1)$ *uar* otherwise with probability $1 - \delta$ v is selected preferentially.
2. With probability q_i following $\boldsymbol{q}$ we select i vertices from $G(t-1)$. The selection policy of these vertices is as follows:
   With probability γ for each of the vertices $\{v^{(1)}, v^{(2)}, \ldots, v^{(i)}\}$ that vertex is chosen *uar* and with probability $1 - \gamma$ the choice is preferential.
3. Add edges $\{\{v, v^{(1)}\}, \ldots, \{v, v^{(i)}\}\}$.

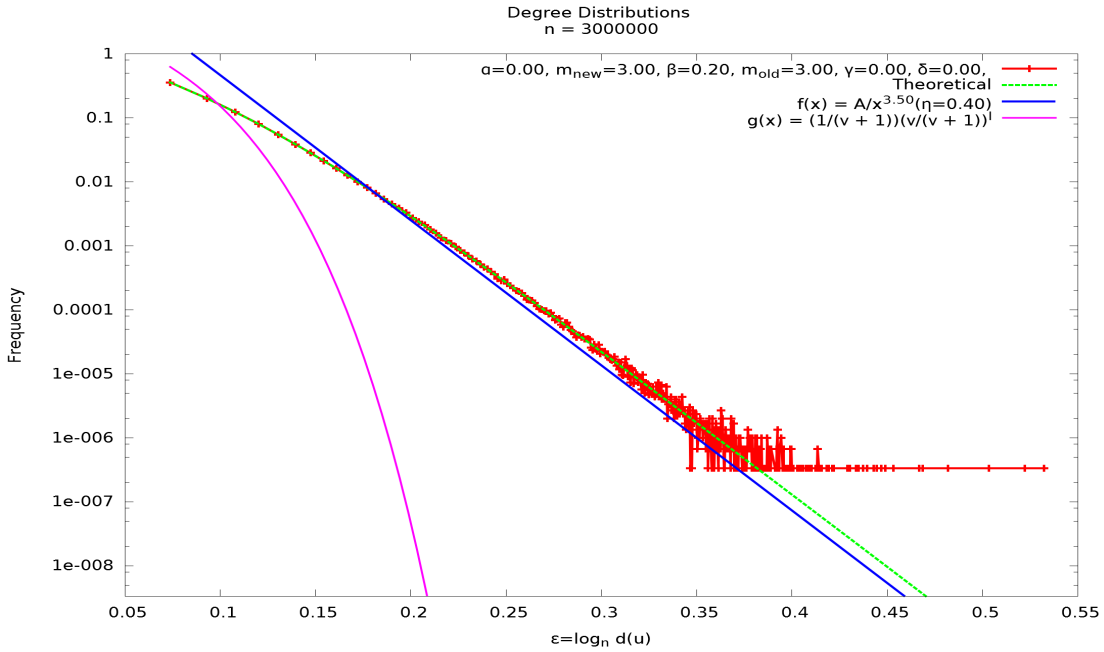


Figure 2.4: Degree distribution for $c = 3.5$

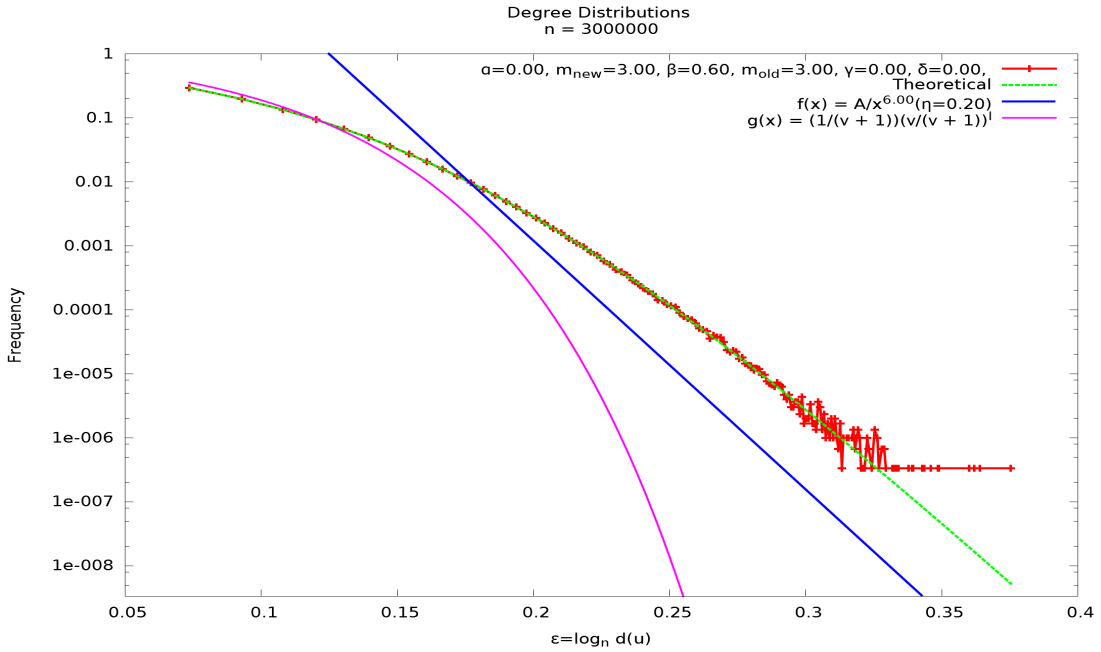


Figure 2.5: Degree distribution for $c = 6$

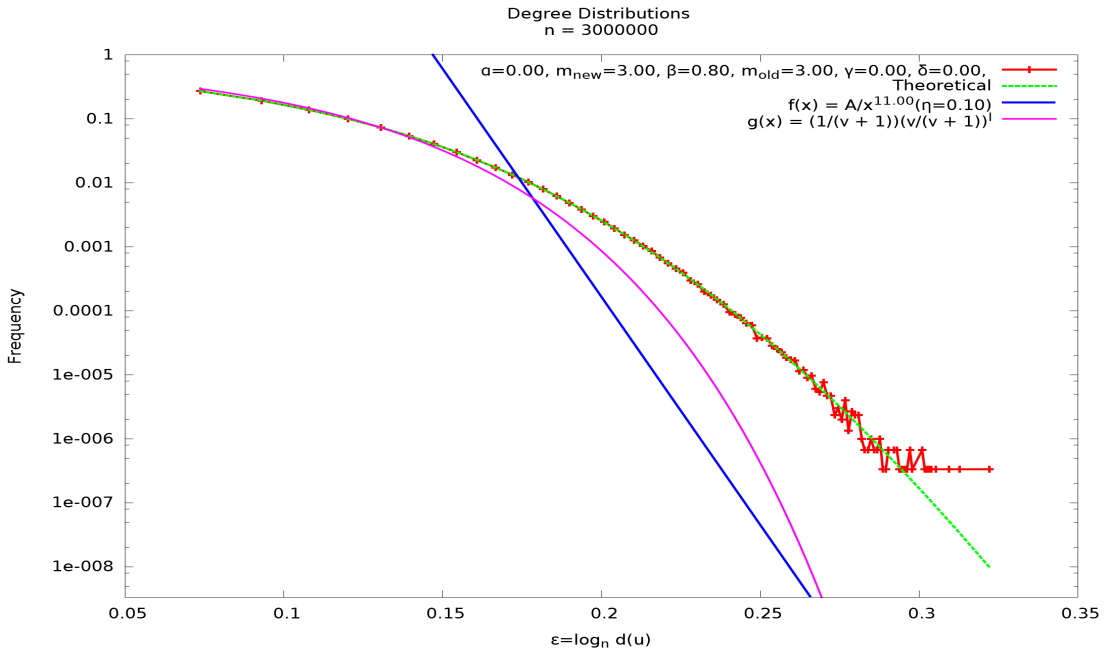


Figure 2.6: Degree distribution for $c = 11$

We refer to this generalized (web-graph) process with power law c as $G(c,t)$. As it is noted above the power-law generated by this process does not depend on the number of edges added per step. In the cases where the number of edges added per step is a constant m then we refer to this graph as $G(c,m,t)$. In [26], Cooper noted the result that the power law c for preferential attachment graphs and web-graphs can be written explicitly as

$$c = 1 + \frac{1}{\eta}, \tag{2.15}$$

where η is the expected proportion of edge end points added preferentially. In the Barabási and Albert model, $\eta = \frac{1}{2}$, as each new edge chooses an existing neighbour vertex preferentially; thus explaining the power law of 3 for this model.

The proportions of preferential edges only depends on the quantities $1-\beta, 1-\delta$ and $1-\gamma$. In the case of the *NEW* procedure each new edge added has 1 preferential endpoint with probability $1-\beta$ therefore, for each added edge the expected number of preferential endpoints is: $(1-\beta)$.

At the same time each new edge added using the *OLD* procedure has 1 preferential endpoint when one of two things happen:

1. The *OLD* vertex is selected preferentially (with probability $1-\delta$) and the other endpoint is selected *uar* (with probability γ).

2. The *OLD* vertex is selected *uar* (with probability δ) and the other endpoint is selected preferentially (with probability $1-\gamma$).

Assuming distributions $P_{NEW}(X_i)$, $P_{OLD}(X_i)$ have a finite mean e.g. $\sum_{i=1}^{\infty} ip_i = \overline{m}$ and $\sum_{i=1}^{\infty} iq_i = \overline{M}$ then the overall η value of this model, as specified in [26] would be:

$$\eta = \frac{(1-\alpha)\overline{m}(1-\beta) + \alpha\overline{M}((1-\gamma)+(1-\delta))}{2((1-\alpha)\overline{m} + \alpha\overline{M})} \tag{2.16}$$

yielding a graph $G(c,t)$ with $c = 1 + \frac{1}{\eta}$.

The BA model can be considered a special case of the General Web-Graph process and can be obtained by setting: $\alpha, \beta = 0$ and $p_m = 1$ where $1 \leq \overline{m} = m < \infty$. The other parameters are not relevant since the *OLD* procedure will never be performed.

The value η occurs naturally in such models in the expression for the expected degree of a vertex. Let $d(s,t)$ denote the degree at step t of the vertex v_s added at step s. The expected value of $d(s,t)$ is given by

$$\mathbf{E}d(s,t) \sim m\left(\frac{t}{s}\right)^{\eta}, \tag{2.17}$$

where η is the parameter defined above (see e.g. [23]). We remind that $f(t) \sim g(t)$ means that $\lim_{t\to\infty} \frac{f(t)}{g(t)} = 1$.

Thus, in the preferential attachment model of [4], $\mathbf{E}d(s,t) \sim m(t/s)^{1/2}$.

The actual value of $d(s,t)$ is not particularly concentrated around $\mathbf{E}d(s,t)$, but the following inequalities are proved in e.g. [23] and [26].The inequalities hold with high probability (**whp**), for all vertices in $G(c,m,t)$. In general, **whp** results mean that, their probability is $1-o(1)$ as $t\to\infty$. $o(1)$ denotes a quantity smaller than any constant.

$$\left(\frac{t}{s}\right)^{\eta(1-\epsilon)} \leq d(s,t) \leq \left(\frac{t}{s}\right)^{\eta} \log^2 t, \tag{2.18}$$

where $\epsilon > 0$ is some arbitrarily small positive constant (e.g. $\epsilon = 0.00001$). The upshot of this, and our reason for explaining this to the reader, is that all vertices v added after step $s \log^{2/\eta+1} t$ have degree $d(v,t) = o\left((t/s)^{\eta}\right)$ **whp**.

Preferential attachment graphs have diameter

$$\text{Diam}(G(m,t)) = O(\log t) \tag{2.19}$$

whp. This was proved for scale-free graphs by Bollobas and Riordan. Detailed investigations of the diameter of preferential attachment graphs have been made by several authors including [14]. Experimental investigations of average distances in the WWW are given in [5, 16].

A crude proof of the $O(\log t)$ upper bound can be made using the expansion properties of the graph. For example, in the preferential attachment graph ($\eta = 1/2, c = 3$) when vertex t is added to $G(m,t)$ the probability that t does not select at least one neighbour in $G(t/2)$ is at most

$$\left(1 - \frac{2m(t/2)}{2mt}\right)^m = \left(\frac{1}{2}\right)^m.$$

Thus $\text{Diam}(G(m,t)) = O(\log t)$ by a 'tracing backwards stochastically' argument. There is a probability of $p = 1 - \left(\frac{1}{2}\right)^m$ that the vertex added at step t, is connected to some vertex added at step $t_u \leq \frac{t}{2}$. Moving backwards, the vertex added at t_u has a probability p of being connected to some vertex added at step $t_v \leq \frac{t_u}{2} \leq \frac{t}{4}$. Therefore, there is a path which exists **whp** starting from u_1 and ending at u_t of length $O(\log t)$, therefore vertices added at steps t and $t-1$ would be at a distance of $O(\log t)$. All vertices added before step $t-1$ would connect to u_1 with even shorter paths. Given a sufficiently large p then we can state that $\text{Diam}\ G(m,t) = O(\log t)$ **whp**.

Experimental results on the web-graph model

Here we will present some experimental results on graphs generated using the web-graph model. The purpose of this section is to show that the analysis found in [24, 26, 28] can be experimentally confirmed in graphs generated using the general web-graph model.

Degree distribution. We note that in [26], the expected degree distribution depends on factors relating to the proportion of preferential and uniform edge endpoints which are added to the graph. These factors are denoted in [26] as ν and η respectively. We can see the value of η under the general web-graph parameters in Equation (2.16). The factor of η is a factor which reflects the proportion of preferential edge endpoints added in the graph. In addition, ν is complementary to η indicating the proportion of edge endpoints added uniformly. The value of ν is given by:

$$\nu = \frac{(1-\alpha)\overline{m}\beta + \alpha\overline{M}(\gamma\delta)}{(1-\alpha)} \tag{2.20}$$

Cooper has shown that the expected number of vertices of degree $m + \ell$ is given by the value of $n_m(\ell)$ shown in the following Equation:

$$n_m(\ell) = \frac{\Gamma(\xi + \frac{1}{\eta})}{\eta\Gamma(\xi)} \frac{\Gamma(\ell + \xi)}{\Gamma(\ell + \xi + 1 + \frac{1}{\eta})} \tag{2.21}$$

where $\xi = \xi(u) = m + \frac{\nu}{\eta}$ and $m = m(u)$ is the degree of vertex u when first added.

In Equation (2.21), Γ denotes the gamma function, the generalization of the factorial function. We remind that $\Gamma(n) = (n-1)!$ if $n \in \mathbb{N}$ and $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\mathrm{d}t$ if $z \in \mathbb{R}$.

The resulting degree distribution from such a graph model can be seen in Figures 2.3-2.6. These figures show graphs generated using the web-graph model. For each of these graphs, $\alpha = 0$, meaning that the *OLD* procedure is never performed. We have varied the parameter β of *uar* attachment in order to obtain graphs with varying values of η, starting from 0.1 and going up to 0.5 (BA model). We remind that $\eta = 0.5$ is the maximum value of η which can occur from only performing the *NEW* procedure. In all these figures the we have used the following distributions for $\boldsymbol{p}$ and $\boldsymbol{q}$: $p_3 = 1$, $q_3 = 1$ and for all $i \neq 3, p_i = 0, q_i = 0$. This means that there would be a constant number of edges added at each step, that number being $m = 3$ in all cases. We have done this to simplify the implementation of the generation model. We also note that the method used to generate the graph whose distribution is seen in Figure 2.3 is essentially the Barabási-Albert generation method and we have obtained this by setting $\alpha = 0$ and

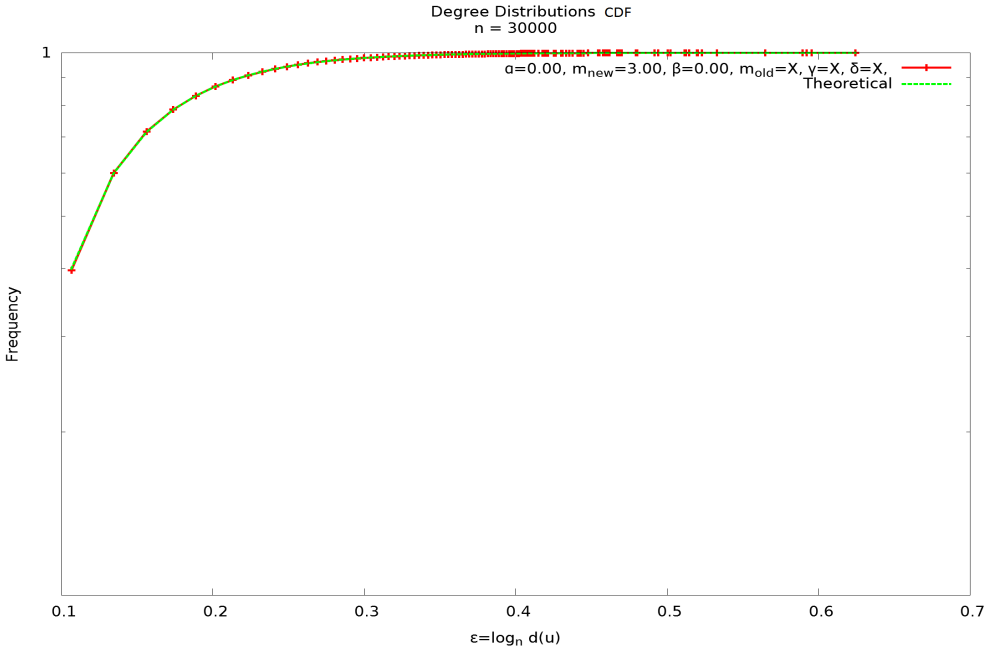


Figure 2.7: $m = 3, c = 3.0$ (Barabási-Albert model)

$\beta = 0$ and using this constant value for m. The additional lines drawn in those plots are due to [26]. We have plotted the expected degree distribution (in the dashed line) from Equation (2.21). In addition we have drawn the asymptotic lines of $f(x) = \frac{A}{x^c}$ and $g(x) = \frac{1}{\nu+1}(\frac{\nu}{\nu+1})^{\ell}$. The reason is to show how the the uniform and preferential portion of the generative model shapes the resulting degree distribution. We note that $g(x)$ is an asymptotic solution to Equation (2.21) when $\eta \to 0$. At the same time $f(x)$ is the power-law to which the degree distribution should converge to for sufficiently large graphs (i.e. when $\ell \to \infty$).

A general observation that can be made is that the rate in which the power-law of the degree distribution seems to converge to the expected theoretical value, however the rate in which this converges seems to be different depending on the parameters used. The slope is much closer to the expected power-law, if the value of ν is low. Overall the actual degree distribution follows the expected distribution precisely, up to a point. Beyond that point there are various explanations on why the degree distribution "breaks". One

reason is that at some point, the $tn_m(\ell) < 1$ i.e. the expected number of vertices of degree $m+\ell$ beyond that point are less than one. At this point, a vertex of degree $m+\ell$ exists with probability $n_m(\ell)$ in which case the frequency of degree $m+\ell$ would be $1/t$ or a multiple of this value (in the unlikely case there are more than 1 vertex of that degree). This explains why the degree distribution seems "flat" towards the high degrees in e.g. Figure 2.3 rather than following the theoretically expected distribution. As a visual example, one can consider the cumulative distribution function (c.d.f.) of the degree distribution rather than the probability distribution function (p.d.f.) to confirm that the experimental values conform the the theoretically expected ones. Such an example can be seen in Figure 2.7.

Maximum degree. The maximum expected degree of a graph generated using the web-graph model would correspond to the vertex with the maximum expected degree. Equation (2.17) gives us an indication of the expected degrees of vertices in such a graph. Since the first vertex introduced, at time $s = 1$, has the maximum degree in the graph, the maximum degree of the graph is expected to be:

$$\mathbf{E}\Delta(G) = \max \mathbf{E}d(u_s, n) \sim mn^{\eta}$$

There is a **whp** bound which is due to A. Flaxman *et al.* [48]. The upper bound we have used was $\Delta(G) \leq t^{\eta} \log^2 t$. In reality, instead of using $\log^2 t$ we could use any function $f(n)$ for which, $f(n) \to \infty$ as $t \to \infty$.

In order to experimentally confirm the maximum degree bound, we have generated graphs with varying values of m and n while keeping $\eta = 0.5$. For each value of n and m we have generated 20 graphs and recorded the maximum degree for each of them. In Figure 2.8 we see the average of these maximum degrees along with the variance across the 20 graphs generated. We observe that the maximum degree is dependent on the parameters m and n. The values of the maximum degree are plotted relatively to the graph size. The relative values of the maximum degree seem to decrease with the graph size, however the rate is very slow. This means that it is unlikely we will be able to

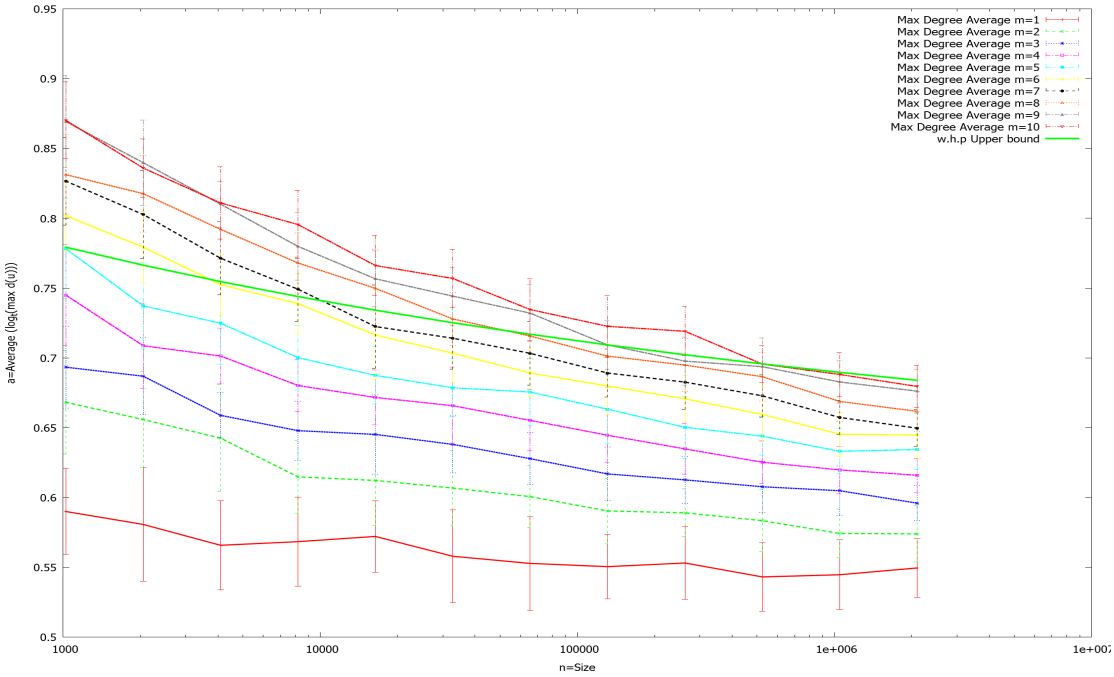


Figure 2.8: Maximum Degree of Web-Graphs with $\eta = 0.5$

experimentally determine the validity of the upper bound for higher values of m due to hardware limitations to the size of the graphs we can generate.

2.2.3 Edge Copying model

Like the preferential attachment, the edge copying model, described by Kleinberg *et al.* [72] produces scale-free graphs. However there is no explicit concept of preferential attachment involved.

This model is a directed model. In [72] the model described only creates vertices with out-degree 1, however, as it is mentioned, it is easy to extend the procedure to out-degree m for each vertex. Here we will describe the model from [72], with the difference that each vertex has out-degree m. The model works as follows:

- We start with an initial graph, typically an m-clique.
- At each step a new vertex v arrives.

- For this vertex v we select a vertex u *uar*.
- As mentioned, vertex u has out-degree m. Assume these edges are $\{(u, w_1), (u, w_2), \ldots, (u, w_m)\}$ We do the following $i = 1 : m$ times:
  1. With probability $1 - \gamma$ we direct an edge from v to w_i.
  2. With probability γ select a vertex v' *uar* and direct an edge from v to v'

This model has been shown to produce power-laws with a coefficient of $1 + \frac{1}{1-\gamma}$ in [72]. Community structure has been observed in the resulting graphs, meaning that e.g. the value of *modularity* (see Section 2.1.7) was higher than the corresponding random or preferential graphs.

In the Section 2.2.2 we have seen the preferential attachment models and noted the effect which the value of η has on the slope of the power-law in these models. We remind that η is the proportion of edge endpoints added preferentially. While in the edge copying model, there is no explicit notion of preferential attachment η can be used as a heuristic to estimate the power-law coefficient [26]. In this case, attaching to a neighbour of vertex u, is preferential in the sense that vertices with higher in-degree are more likely to be selected. This would mean the value of η would be $1 - \gamma$, which confirms the power-law coefficient of [72].

The resulting degree distribution from the above generation model can be seen in Figure 2.9. The slope in this case confirms the theoretically expected and is $c = 3$.

2.2.4 Triangle Closing Model

This model was created based on the observation that most links in OLSNs are created locally. A new link typically connects vertices which were previously no more than 2 hops apart [85]. There are several variations of this model discussed but they all follow this basic idea: At each step a new vertex v arrives. For this vertex v we select a vertex u *uar* and direct an edge to that vertex. Having done this then we proceed to create as

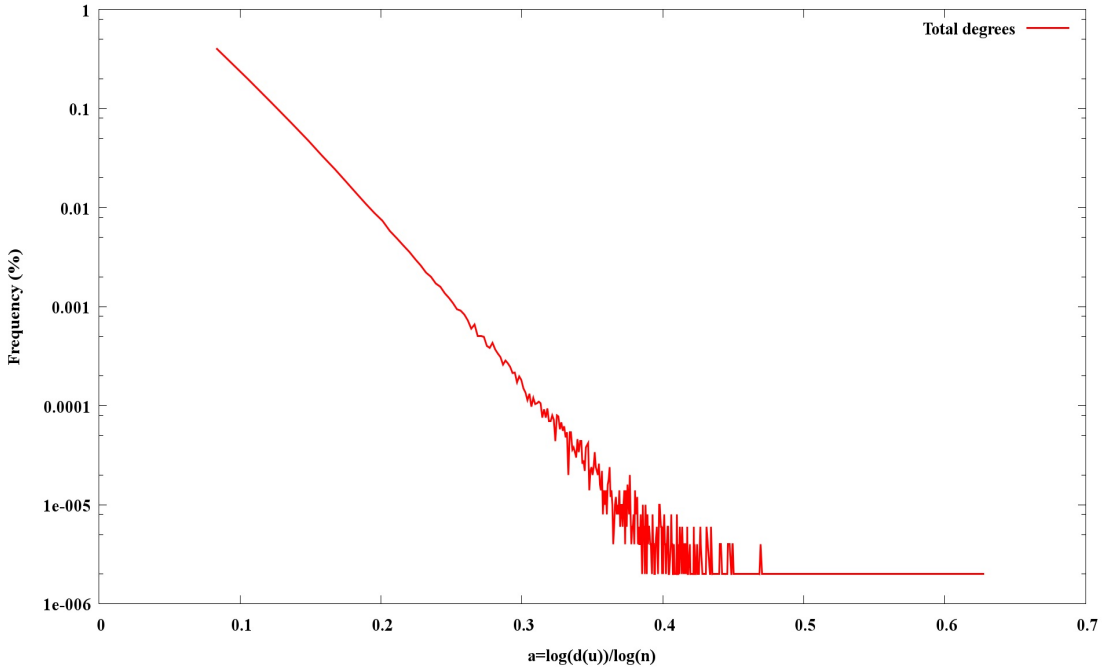
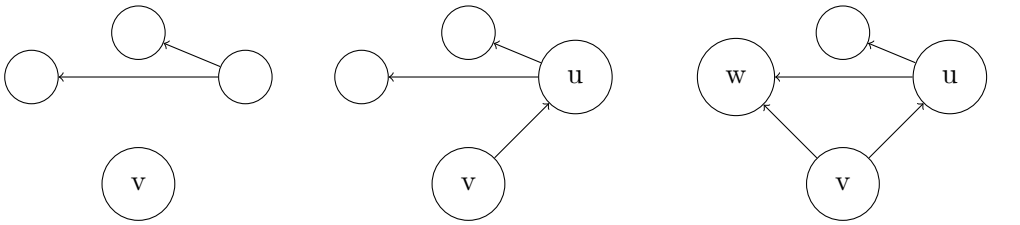


Figure 2.9: Degree distribution of the edge copying graph with $\gamma = 0.5$

many additional edges as we wish, by using a certain selection policy to select vertices of distance 2 from v. Typically we would select a random neighbour u of v and then select a random neighbour w of u and direct and edge from v to w. Initially v would only have 1 neighbour, but during subsequent selections the newly added neighbours can also be selected. An illustration of this procedure can be seen in Figure 2.10.

The way the vertex is selected depends on a given policy and there are several policies mentioned in [85]. These policies were:



Vertex v is added | Select vertex u uniformly and direct edge | Select w and direct edge

Figure 2.10: Triangle closing

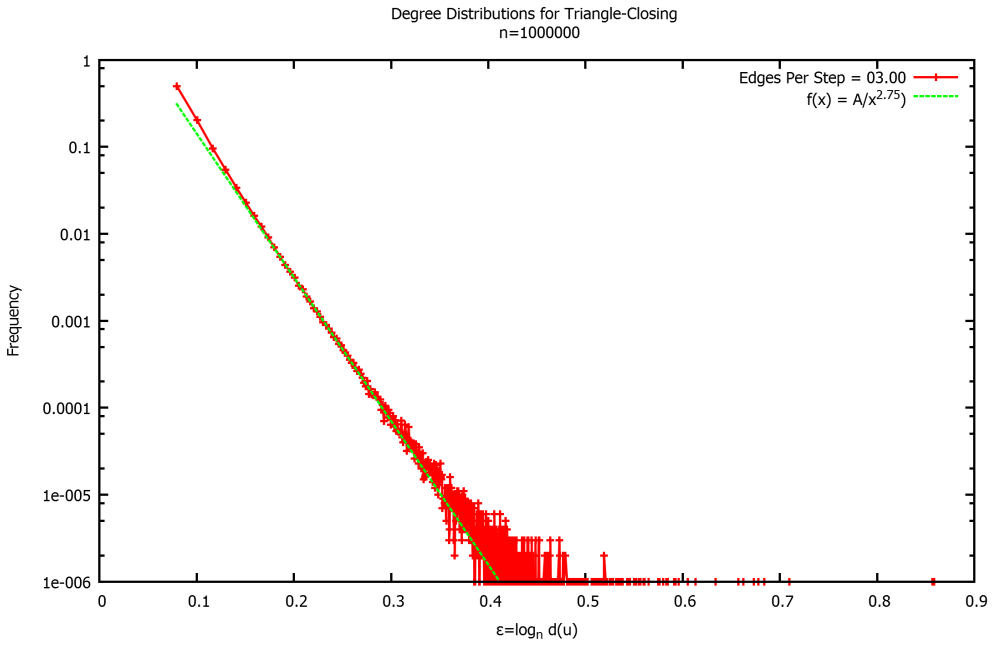


Figure 2.11: Degree distribution of the triangle closing graph with random-random selection policy

1. **Random.** The vertex is selected uniformly at random for the set of neighbours.
2. **Preferential.** The vertex is selected preferentially (proportionally to degree) among the neighbours.
3. **Common.** The vertex is selected proportionally to the number of common neighbours it shares with v.
4. **Activity.** The vertex is selected proportionally to the number of steps since it has been last selected.

Any combination of the above policies can be applied, once for selecting a neighbour of v and then a second time for selecting a neighbour of that neighbour.

The generated graphs were compared to some real OLSNs with respect to how each vertex directs edges to other vertices. The comparison was done by observing the evolution of some real OLSNs graphs, and then using a maximum likelihood estimator to get the likelihood of the same process being repeated by the generative model. It was suggested

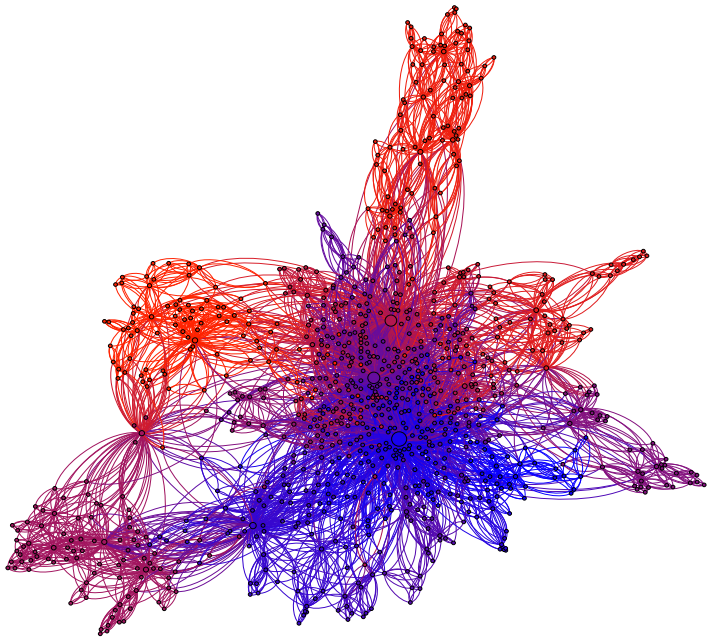


Figure 2.12: Triangle closing graph, random-random selection policy. $n = 1000, m = 3$

by Leskovec that the random neighbour of random neighbour policy is the simplest one and worked surprisingly well. While other policies, such as most active neighbour of most active neighbour, may work better the increase in accuracy is marginal [85].

We have generated a graph based on this model with parameters $n = 10^7$ and number of edges added per step being 3. The selection policy we have used was *Random–Random* i.e. random neighbour of a random neighbour. The resulting degree distribution from the above generation model can be seen in Figure 2.11 where the slope of the power-law is approximately $c = 2.75$. In this case what is plotted is the total degree frequency (the out-degree is m for all vertices). In addition, we have observed a community structure in these graphs, which can be seen in Figure 2.12. Each detected community was given a different colour for differentiation.

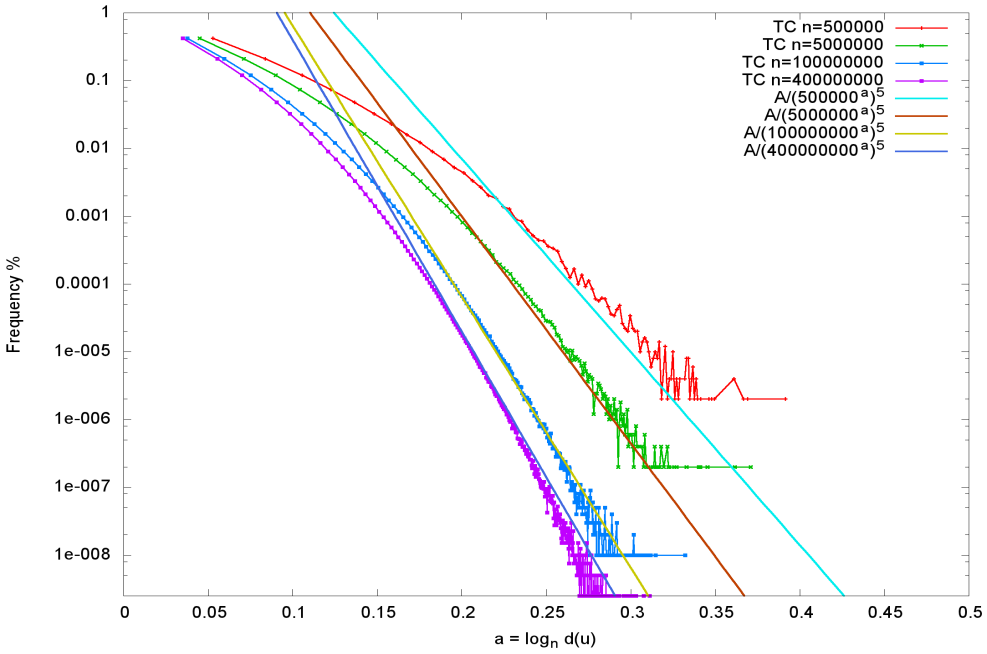


Figure 2.13: Undirected triangle closing slope. Convergence rate to c.

Undirected triangle closing model

In this section we will present an undirected variation of the triangle closing model. Specifically, at each step we add vertex v to the graph. We then and select a vertex u *uar* and add edge (u, v). We then select vertex w from $N(u)$ and create an edge (v, w). This forms a triangle between vertices u, v and w. We remind that according to [26], the slope of the power-law would converge to $c = 1 + \frac{1}{\eta}$. In this case $\eta = 0.25$, since $1/4$ endpoints are added preferentially, and therefore $c = 5$. In addition we expect the maximum degree to be $\Delta(G) = O(n^{1/4})$. While the model is difficult to analyse formally, the heuristic calculation of c seems to be fairly accurate. We have experimentally confirmed that the slope does seem to converge to $c = 5$. The maximum degree also seems to converge to $\Delta(G) = O(n^{1/4})$. This result can be seen in Figure 2.13

In general, we have experimentally determined that for many other undirected graph generation models, the heuristic $c = 1 + \frac{1}{\eta}$ does give a good estimate of the slope of the power-law, however the rate at which the slope converges to c is unknown.

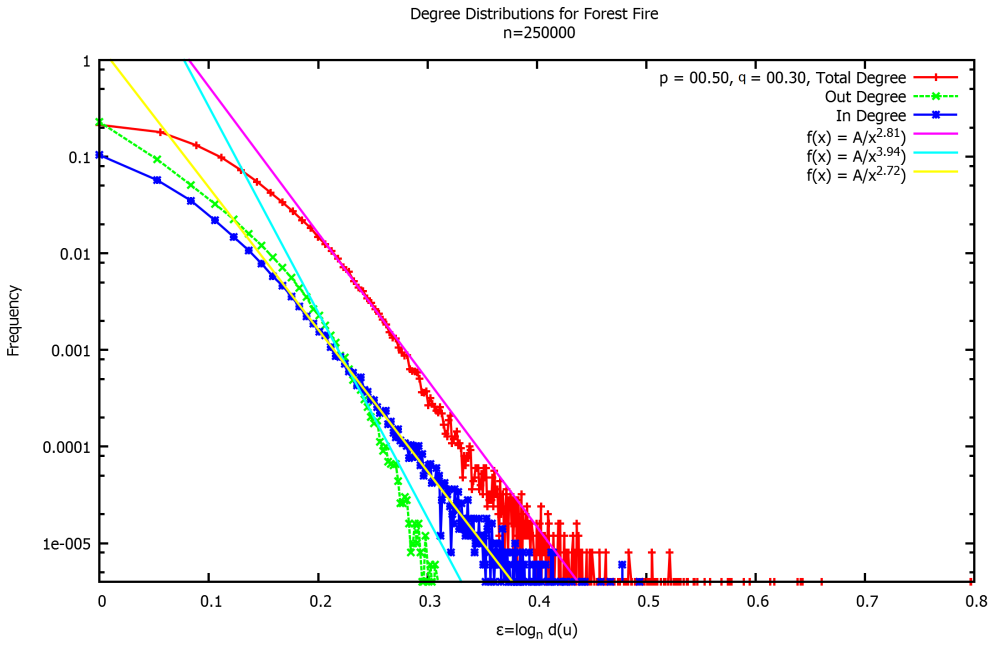


Figure 2.14: Degree distribution of the forest fire model with $n = 2.5 * 10^6 p = 0.5, r = 0.3$

2.2.5 Forest Fire Model

This model, was proposed by Leskovec *et al.* [88] as an intuitive model on how OLSNs evolve over time. It is very similar to the triangle closing model when viewed from the perspective of the locality of new links. Like the triangle closing model, the forest fire model is also a directed model, but unlike the triangle closing model, this model generates graphs with power-law degree distributions in both the in-degrees and out-degrees. A community structure is also observed. Moreover, according to [88] the edge densification power-law holds for each step of the generative process. In addition, the diameter of graphs generated using this model decreases as the graph size increases [88]. An evolving graph, such as an OLSNs or evolving graph generation process, is said to have an edge densification power-law if $E(t)$, the number of edges at time t is proportional to some power of the number of vertices $V(t)$, i.e. $E(t) \propto V(t)^{\alpha}$ with $\alpha > 1$. This phenomenon has been observed by Leskovec *et al.* [68] in some OLSNs.

The forest fire model requires 2 parameters: p, q. The generative method is as follows:

1. At each step we add a new vertex v and direct an edges to a vertex u selected *uar*. This vertex is referred to as the *ambassador* of v ([88]).

2. We generate two random numbers x and y, which are geometrically distributed, with means $\mathbf{E}x = \frac{p}{1-p}$ and $\mathbf{E}y = \frac{pq}{1-pq}$ respectively. We select x out-edges of u, i.e. $\{(u, w_1), \ldots, (u, w_x)\}$ and y in-edges of u i.e. $\{(z_1, u), \ldots, (z_y, u)\}$. We then direct edges from v to all vertices w_i and from all vertices z_i to v.

3. For each edge (v, w) created, step (2) is repeated.

4. We continue until no new edges are added at which point we add a new vertex and repeat the above process.

This method has a very good intuition of why it should generate graphs which look like OLSNs from a sociological point of view, and in practice it does generate such graphs under a certain condition. Due to the complexity of the method it has not yet been formally analysed. There are still uncertainties on why the graphs generated look the way they do and how the input parameters (p, q) affect the properties of the generated graph. For a range of those parameters the graph may tend to become a complete graph where by definition the desired properties are not met i.e. there is no power-law in the degree distribution and the graph is not sparse.

Additionally Leskovec has shown that there is a specific range of parameters that he called the *"sweet spot"*. In this range of parameters, the generated graphs have the properties described, i.e. power-law degree distribution on both in and out-degrees, community structure, small-world diameter and edge densification. The degree distribution of a graph generated using this method can be seen in Figure 2.14. The specific graph generated has a power-law degree distribution, with slopes being $c = 2.8$, $c_{out} = 3.9$ and $c_{in} = 2.7$.

2.2.6 Stochastic Kronecker Graph

This method as a generation model was proposed by Leskovec *et al.* [87]. It takes advantage of the self-similar structure observed in many real world graphs and uses the Kronecker matrix multiplication (a form of tensor product applied on matrices) in order to generate graphs.

This method itself requires a good initiator matrix to be chosen and depending on that, the properties of any size of graph generated can be calculated using the properties of the initiator matrix product. However choosing an appropriate initiator matrix is still an open problem and thus, this method is not as effective for generating graphs as it could potentially be.

The major advantage and use of this method however is not to generate graphs. It is used to find which initiator matrix could be used to produce graph which is similar to a given graph. It was suggested by Leskovec that finding a good initiator which is most likely to have produced a given graph and then applying the Kronecker multiplication will produce a graph very similar to the original graph. This can be used to model the given network either at a scale or determine how it may look like when it grows.

The problem of finding the initiator, while generally an NP-complete problem, was proven to be solvable by approximation in $O(n)$ time [86], and this may prove to be a valuable tool in analysing networks and their temporal evolution.

The Kronecker product is an operation of two matrices of arbitrary size resulting in a block matrix. This operation, which we now describe, is completely unrelated to the normal matrix product.

Assume we have two matrices M_A and M_B of dimensions $m \times n$ and $p \times q$ respectively.

$$M_A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \qquad M_B = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,q} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ b_{p,1} & b_{p,2} & \cdots & b_{p,q} \end{pmatrix}$$

The Kronecker Product of these matrices is symbolised as:

$$M_A \otimes M_B$$

and is a block operation defined as:

$$M_A \otimes M_B = \begin{pmatrix} a_{1,1}M_B & a_{1,2}M_B & \cdots & a_{1,n}M_B \\ a_{2,1}M_B & a_{2,2}M_B & \cdots & a_{2,n}M_B \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1}M_B & a_{m,2}M_B & \cdots & a_{m,n}M_B \end{pmatrix}$$

In the case of the Stochastic Kronecker Graph we require that $M_A = M_B$, $m = n$ and $0 \leq a_{ij} \leq 1$. We will symbolize $M_A \otimes M_A$ as $M_A^{(2)}$ which would contain elements $a_{ij}^{(2)}$. Further analysis of this model was done by M. Mahdian *et al.* [96]. In [96] a stochastic Kronecker graph with initiator matrix of size 2 is defined by:

1. An integer k
2. A symmetric 2×2 matrix ϑ with entries $\vartheta(1,1) = \alpha$, $\vartheta(1,2) = \vartheta(2,1) = \beta$, $\vartheta(2,2) = \gamma$ where $0 \leq \gamma \leq \beta \leq \alpha \leq 1$.
3. The graph has $n = 2^k$ vertices

Specifically, in [96] it was shown that there are transitional phases for the emergence of a giant component and for the connectivity. Initially the graph is not expected to be connected and all the disconnected components are relatively small. The aforementioned

transitional phase is a size threshold in which the majority of the small components connect to each other, creating a giant component which makes the graph nearly connected. Additionally Mahdian proved that the diameter beyond the connectivity threshold is constant.

In general the matrix ϑ does not need to be symmetric and in [86], a non symmetric matrix was used to simulate directed OLSNs. The matrix $M_{\vartheta}^{(k)}$ or simply $\vartheta^{(k)}$ is called the "adjacency probability matrix" of a graph. In order to generate the actual graph which results from the k-th Kronecker multiplication we create a graph where for each vertex pair u_i, u_j the probability that there is an edge $e_{ij} = (u_i, u_j)$ is $P(e_{ij}) = a_{ij}^{(k)}$.

A major drawback of this approach is that the calculation of the k-th power Kronecker power of the initiator matrix ϑ takes $O(2^{2k}) = O(n^2)$ time and $O(n^2)$ space. This is due to the fact that, since the initiator matrix (typically) does not contain zero elements, the Kronecker product would also have the same property. It is therefore not feasible to generate large graphs using this approach. Due to this, Leskovec *et al.* [86] have suggested an alternative way to obtain graphs generated via the k-th Kronecker multiplication though a fast generation procedure. This is based on the following ideas:

1. Since the existence of each edge is an independent Bernoulli trial with success probability $P(e_{ij})$. The expected number of edges would be $\mathbf{E}\ m = \sum_{i=1}^{n}\sum_{j=1}^{n} P(e_{ij})$ with variance $\mathbf{Var}\ m = \sum_{i=1}^{n}\sum_{j=1}^{n} P(e_{ij})(1 - P(e_{ij}))$.

2. From the central limit theorem, m would follow a normal distribution with mean $\mathbf{E}\ m$ and variance $\mathbf{Var}\ m$.

3. The expected number of edges $\mathbf{E}\ \ m$ relate to the initiator matrix since $\sum_{i=1}^{n}\sum_{j=1}^{n} P(e_{ij}) \approx \left(\sum_{i=1}^{k}\sum_{j=1}^{k} \vartheta(i,j)\right)^k$.

4. Due to the self-similar structure of $\vartheta^{(k)}$, we can add edge (i, j) with probability $P(e_{ij})$ by generating $i = 1 : k$ recursive coordinate samples from matrix ϑ. Each sampled coordinate (u, v) would be according to the probability $\frac{\vartheta(u,v)}{\sum_{i=1}^{k}\sum_{j=1}^{k}\vartheta(i,j)}$.

5. The i-th sampled coordinates would correspond to the block coordinates of $\vartheta^{(k)}$ when converted to a block matrix with blocks of size $2^{(k-i)}$. By following each block recursively we eventually sample a coordinates from matrix $\vartheta^{(k)}$ with probability $P(e_{ij})$.

Given parameters ϑ and k, the exact generative process is as follows:

- We generate a number r following a normal distribution with mean $\mathbf{E}\ m$ and variance $\mathbf{Var}\ m$.
- We sample r coordinates e.g. (u, v) according to their probability $P(e_{uv})$ of matrix $\vartheta^{(k)}$.
- We create an edge between u and v.

The drawback of this approach is that the the variance of the Bernoulli trials becomes too large when p approaches 0 or 1. In this case there graph generated with this approach would may not be as likely to have been generated using the Kronecker multiplication. In general this approach makes this generation method feasible since the graphs generated would generally be sparse and there would be no need to maintain the entire stochastic matrix $\vartheta^{(k)}$ in memory.

2.2.7 Affiliation Networks

In the work of S. Lattanzi *et al.* [78] it is claimed that all the theoretical understanding of pre-existing graph generation models failed to explain properties which were recently observed in social networks [68, 87]. They propose a new method which is based on previous work on bipartite models of social networks. This model aims to "capture the affiliation of agents to societies"

In this model there are two distinct graphs:

- A bipartite graph that represents the affiliation network, which in [78] is referred to as $B(Q, U)$

- The social network graph which in [78] is referred to as $G(Q, E)$

Specifically, the model is based on the concurrent evolution of the two graphs, B and G. The following parameters are required: β, c_Q, c_U, s. At each step one of two events may occur:

1. With probability β: Evolution of Q.
2. With probability $1 - \beta$: Evolution of U.

The evolution of Q involves the following actions:

- A new vertex q arrives and is added to Q.
- A vertex q' is selected from Q *preferentially* and q connects to c_Q neighbours of q' in $B(Q, U)$ selected *uar*.
- Vertex q connects to vertex $q'' \in G$ if they have at least one common neighbour in U.
- There are s vertices $S = \{q_1, q_2, \ldots, q_s\}$ selected from preferentially Q , and the edges $\{(q, q_1), \ldots, (q, q_s)\}$ are added.

The evolution of U involves the following actions:

- A new vertex u arrives and is added to U.
- A vertex u' is selected from U *preferentially* and u connects to c_U neighbours of u' in $B(Q, U)$ selected *uar*.
- Vertices q_1 and q_2 are connected in G if u is a neighbour to both of them.

In the above case the set Q is shared among both graphs. Their intuition in creating this model is based on observations on social phenomena in online graphs (such as the citation network). In their example they make use of the citation network and in this

context Q is the set of papers and U the set of topics those papers are about. When a new paper emerges it is likely to be based on an older paper, referred to as the *prototype* and it is also likely that focus on (a subset of) the topics in which the *prototype* focuses on. Based on this, new vertices emerging in Q will consist of an edge copying flavour as described in Section 2.2.3. In a similar fashion when a new topic emerges it is likely to be based, or inspired from, an existing topic.

Based on the above example the graph $G(Q, E)$ is constructed when taking into consideration that when an author adds references to a new paper that author will cite most or all of the papers on that topic and some papers of general interest. It is mentioned that this intuition can be applied to other social graphs as well and we can assume within this statement that we can consider Q to be a set of interests a person may have and $G(Q, E)$ to be a social interaction graph between people. It is more likely for people who share the same interest to be connected in G and in addition people without common interests may be connected as a result of the popularity of one or both of those people.

From this intuitive understanding there are two factors that emerge which are an edge copying flavour and a flavour of preferential attachment based on degree. In fact the suggested model contains both these elements in some way. In fact since graph $B(Q, U)$ uses the edge copying mechanism heavily it does exhibit a power-law degree distribution and a community structure as it is proven in the paper. The graph $G(Q, E)$ which also includes the degree preferential attachment mechanism as well as common affiliation links between vertices of Q also exhibits this phenomenon. In addition this model claims a densification power-law and bounded diameter.

This model is worth noting due to the fact that all the above properties are properties which have been observed in most online graphs (social, citation, Peer-To-Peer (P2P) etc.) but more importantly this model provides a proven power-law degree distribution, densification and bounded diameter.

2.2.8 Random Walk Graph

In this section we describe a simple model based on RWs. The Random Walk (RW) graph model is a graph generation model which we have created in order to generate graphs in which each edge is created with a clear intuition, that is a model which simulates a user searching through the network by going through local links, to decide who to connect to. In addition since this is a graph generation model which is based around a SRW, it is easy to understand and we can make use of RW properties to further extend and generalize it. The main reason we have devised this model is that the area of RWs is one we use extensively. There are many reasons why RWs fit well in OLSN analysis, which we discuss further in Chapter 5.

At each step the method creates a new vertex u and attaches to a random existing vertex v in the graph. Then it performs a SRW of s steps starting from u and stores all the visited vertices in a list L allowing repeated entries. After the end of the walk, a vertex v is selected from L and an edge (u, v) created. This process may be repeated up to $m - 1$ times to add m total edges per step. There are many possible ways to select v, including (but not limited to) a random selection or preferential selection. In general this model is easily extendible by using other RWs instead of a SRW.

Ideally this should generate power-laws since it does have both growth and some flavour of preferential attachment, i.e., it combines both Model A and Model B mentioned in Section 2.2.2. Details on why there is preferential attachment can be further found in Chapter 3 where RWs are analysed in depth. We will mention here that for $s > \tau_2$ (see Section 3.1.3) the expected number of times an undirected SRW visits vertex v is approximately $(s - \tau_2)\frac{d(v)}{2m} + O(1)$. The $O(1)$ term would be the expected number of times v is visited in steps less than τ_2. This would mean in general that vertices with higher degree are visited more often and therefore have a higher probability of being selected as edge endpoints. This by definition is a preferential vertex selection. We note that τ_2 is referred to as the "mixing time" of a random walk, and it is the number of steps required before the visited vertex stops being correlated to the starting vertex. If

$s \gg \tau_2$ then $(s - \tau_2)\frac{d(v)}{2m} + O(1) \approx s\frac{d(v)}{2m}$. If instead of having a fixed stopping time s we run the random walk until it has covered the entire graph, we have a model which is identical to preferential attachment. On the other hand if we stop the walk at time $s \approx \tau_2$ we have a model similar to triangle closing (described in Section 2.2.4) but rather than triangles, we insert cycles of length at most s since all the vertices visited by the RW would be at a distance at most s.

This model also generates a good community structure for small s. This is due to the fact that chosen vertices are at most s steps away from the newly added vertex. There are two different variations of the model we consider, which is simply using either undirected random walks (i.e. ignoring edge direction) or using directed random walks and walking only along out-edges. The power-law coefficient varies with the parameters is typically $c > 3$. The model is directed but only presents with power-laws in the in direction, which by design is what it is intended to do. It is unclear on how the parameters affect it therefore it may be a good candidate model but needs further analysis and modifications.

We have generated a large number of such graphs to compare how the parameters affect the resulting graph properties. Results of these comparisons can be seen in Figures 2.15-2.20. In particular we have varied the parameters of m (edges per step), s (number of steps per random walk), and whether the random walk followed the directed graph or ignored edge direction. We note that values of $m = 1$ imply that no random walk is performed and the resulting distribution would be that of a growing random graph generation process (Model A, Section 2.2.2).

We have determined that the number of steps in both variations do not seem to affect the generated power-law coefficient. We have only experimented with small values of s, i.e. $s \leq 100$. However, the number of steps does affect the modularity Q of the graph, which, when m was kept constant ($m = 3$) was determined to vary from $Q = 0.5$ for $s = 10$ to $Q = 0.3$ for $s = 100$. Furthermore what does affect the power-law coefficient is m which larger m means smaller coefficients. Additionally m seems to affect Q which

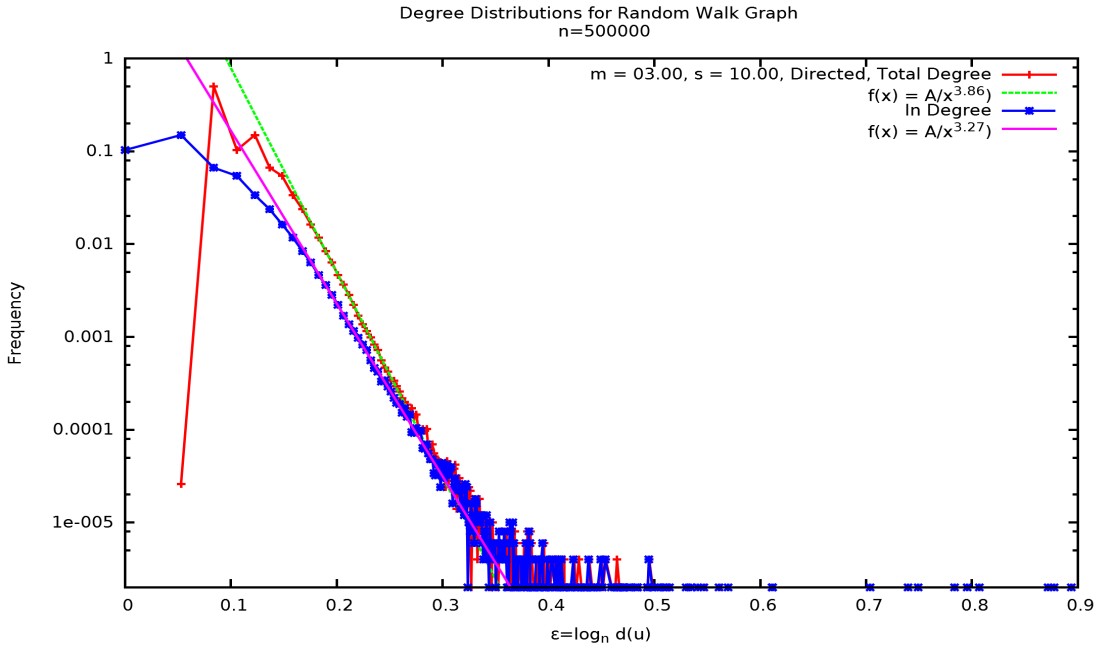


Figure 2.15: Directed RW. $n = 150000$, $c = 3.86$, $c_{in} = 3.27$

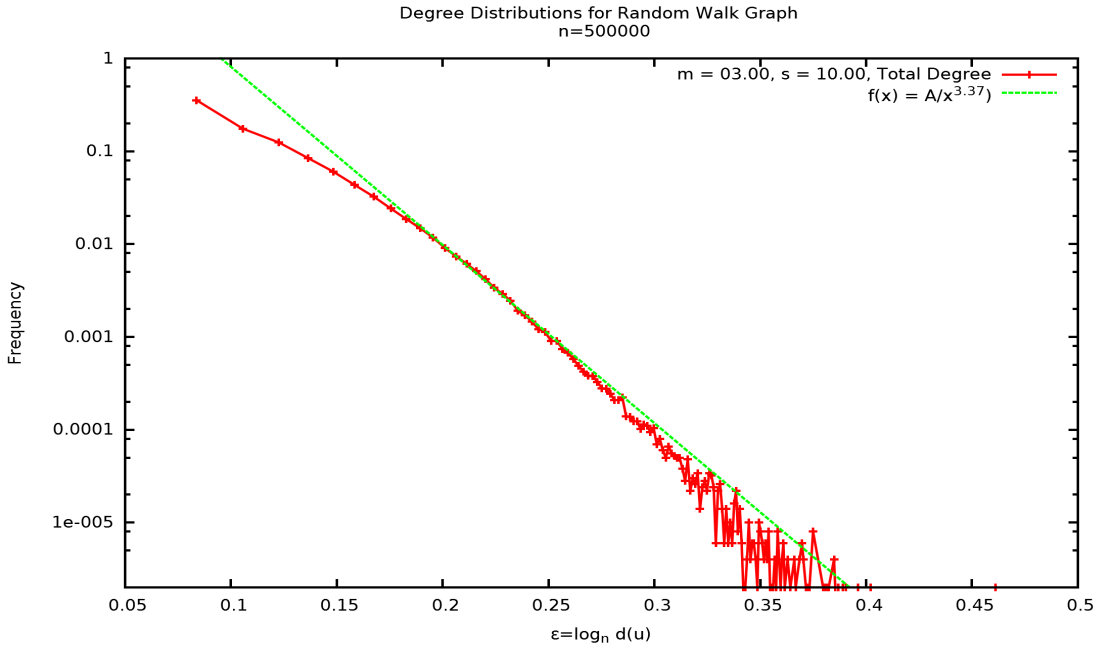


Figure 2.16: Undirected RW. $n = 150000$, $c = 3.37$

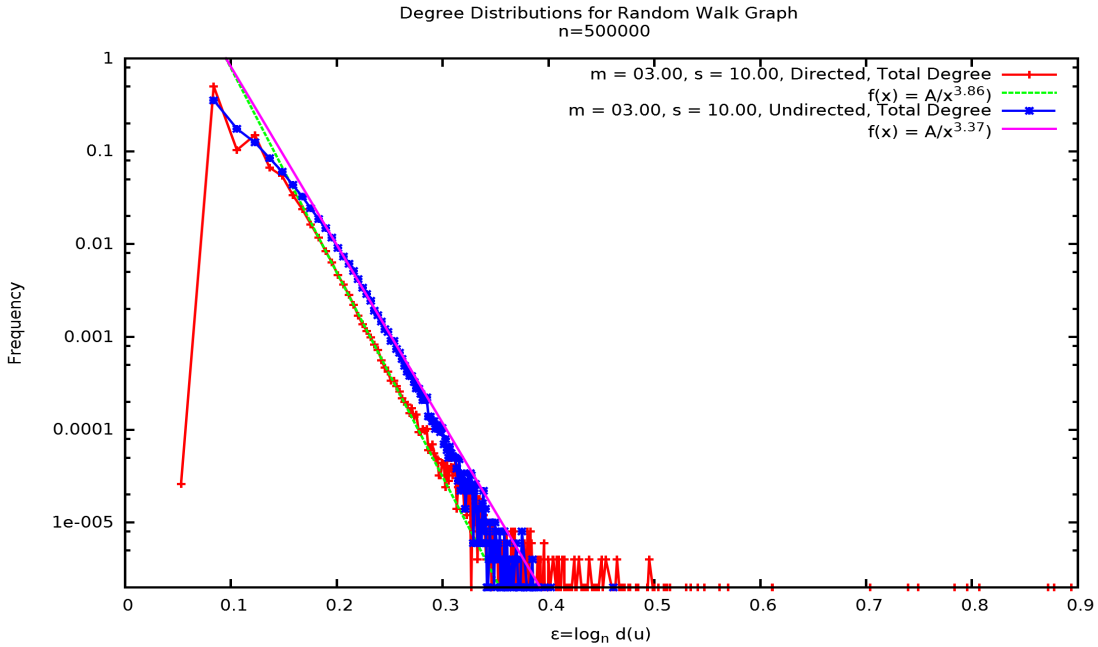


Figure 2.17: Directed vs Undirected Total Degrees, $n = 150000$

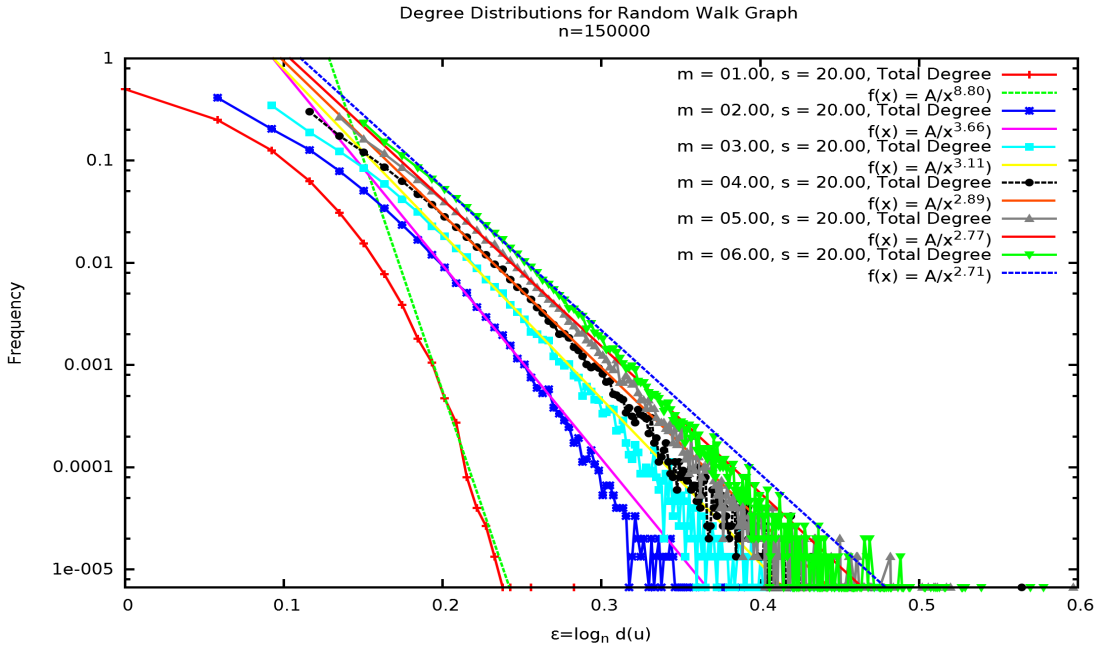


Figure 2.18: Undirected. Constant $s = 20$ Varying m. $n = 150000$

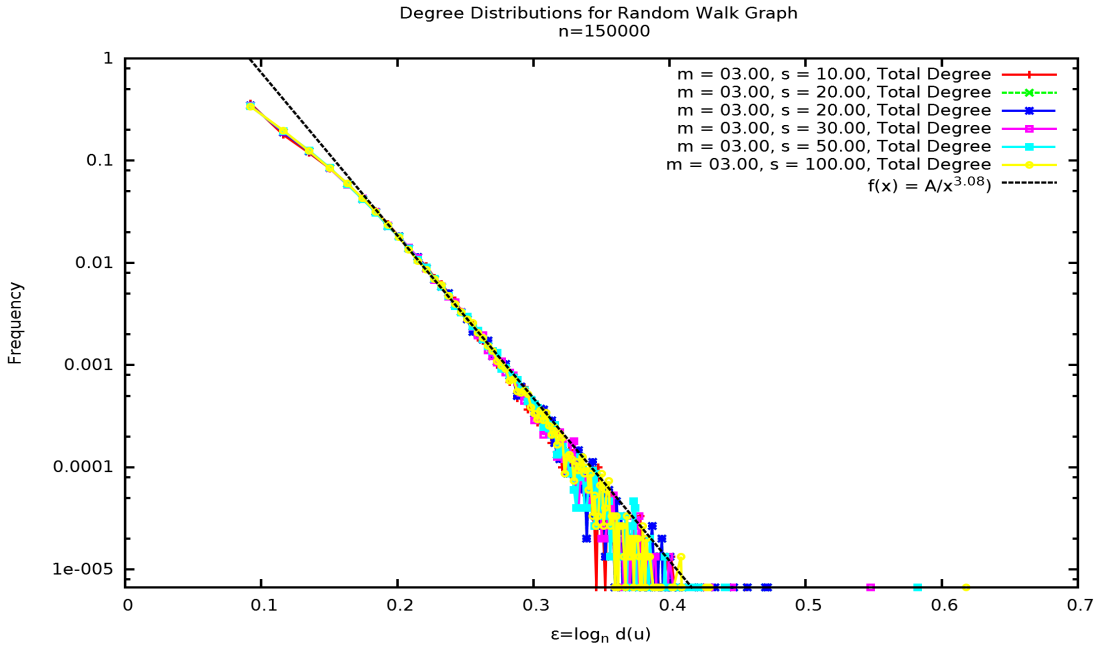


Figure 2.19: Undirected. Constant $m = 3$ Varying s. $n = 150000$

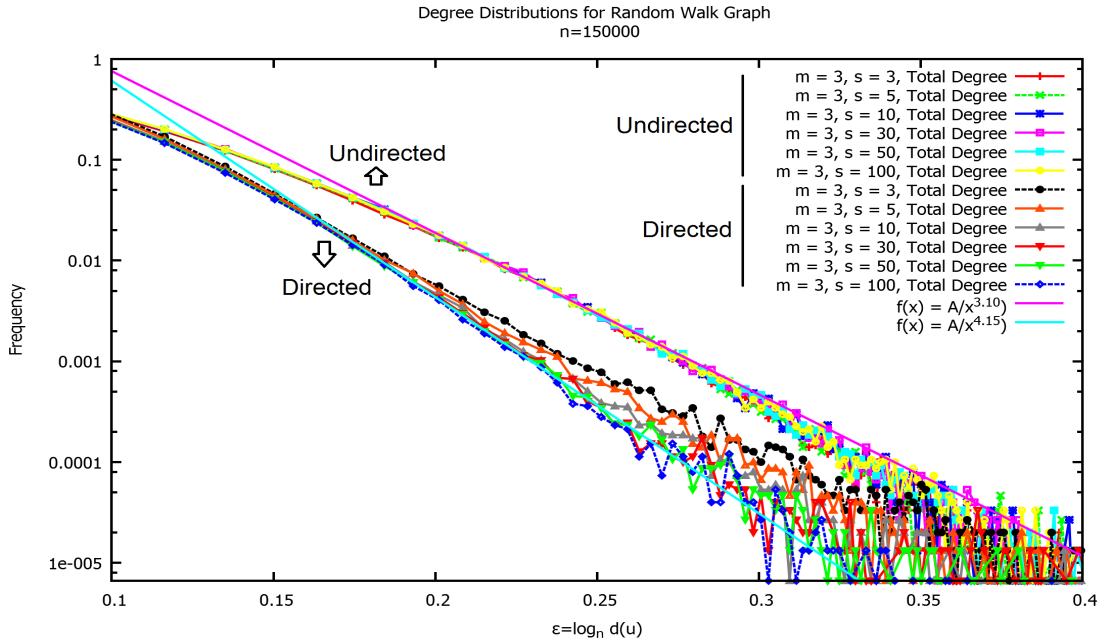


Figure 2.20: Varying s, Directed vs Undirected.

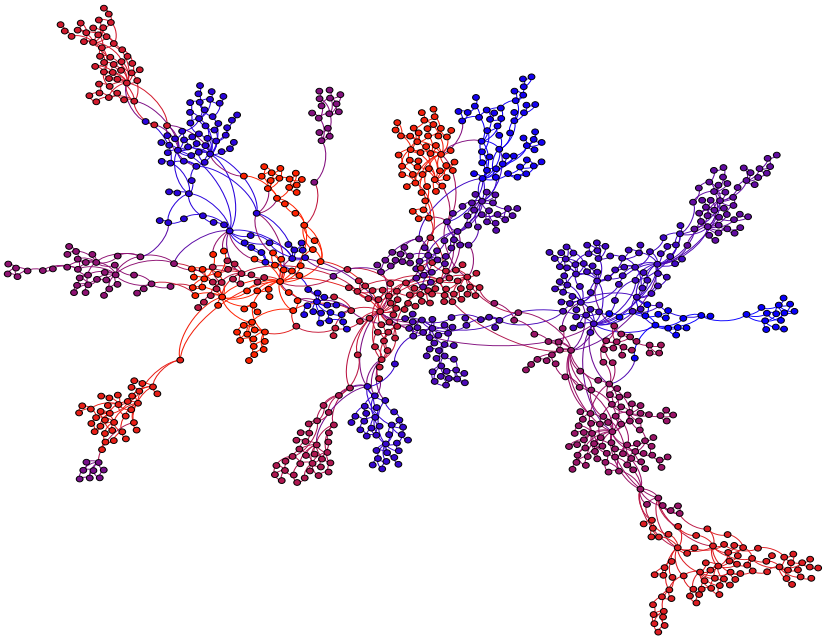


Figure 2.21: Community structure of undirected random walk graph. $n = 1000, m = 2, s = 3$

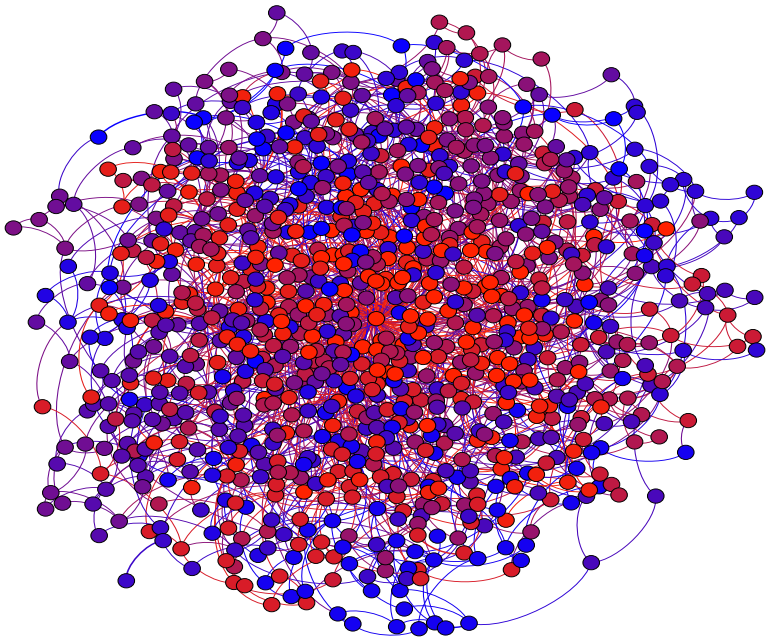


Figure 2.22: Community structure of undirected random walk graph. $n = 1000, m = 2, s = 50$

when we kept s constant ($s = 50$) Q appears to drop from $Q = 0.6$ for $m = 2$ down to $Q = 0.25$ for $m = 5$. The results are generally similar for directed and undirected random walks, with the difference that when the walks are directed the power-law coefficient is generally higher than that of the graph generated with the same m and s but by using undirected random walks. A comparison of the resulting community structure can be seen in Figures 2.21 and 2.22. Specifically each detected community has been given a different colour. We can observe in Figure 2.21, the communities seem better defined, and this is reflected by the modularity score Q which in this case was 0.8. This is not the case in Figure 2.22 where there seems to be overlap in the community structure and $Q = 0.5$. We note that s also affects the diameter of the generated graphs, with the graph in Figure 2.21 with $s = 3$ having a diameter of 18 while the graph in Figure 2.22 with $s = 50$ having a diameter of 8. These results are expected since as s increases, then the probability of u connecting to a vertex of high distance increases.

2.2.9 Preferential attachment and message propagation

This model is essentially a combination of the well known preferential attachment model with an intuitive flavour of the Twitter link creation procedure which we have added. It is a model we devised based on the idea that there are two ways of creating links:

1. A new vertex joins and connects to m other vertices with probability proportional to each vertex's total degree (normal preferential attachment).

2. An existing vertex activates and begins transmitting a message down to its out-edges. Each recipient of that message has a probability p to retransmit that message. After the process has died out (or reached a maximum number of hops) then from all the vertices which retransmitted, m of them choose to create a directed link to the originator of the message (message propagation). We chose only m links to be created during each step to ensure that the graph remains relatively sparse.

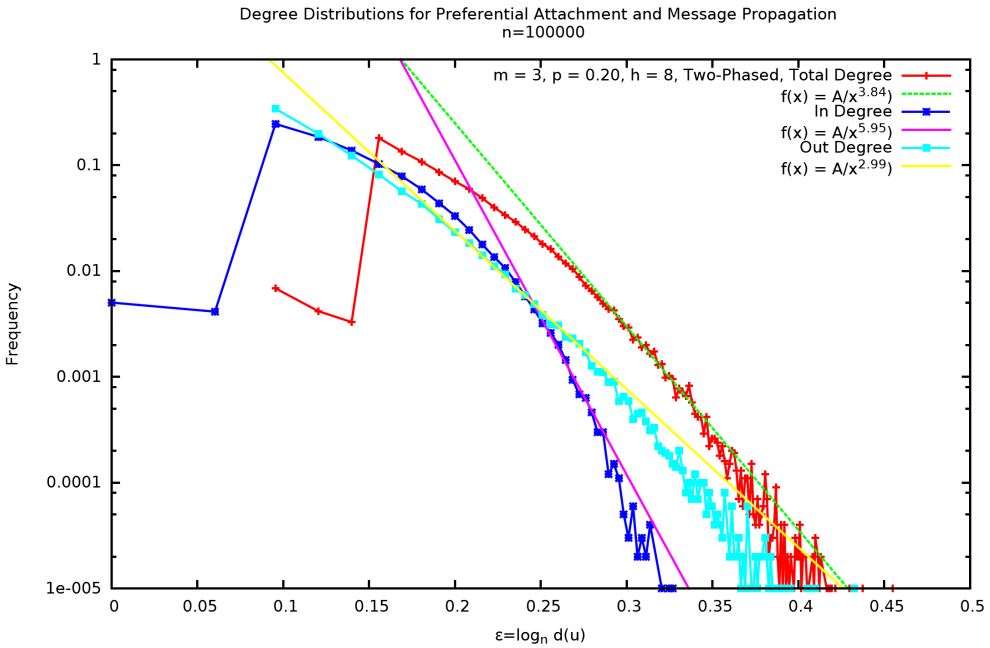


Figure 2.23: Preferential Message Propagation

It is a directed model, and specifically the preferential attachment portion of the model is preferential on the in-degrees, since we are always creating m out-edges when performing a preferential attachment step.

While there are multiple ways to generate graphs which combine these two procedures, we will only consider one here. This variation of the model requires parameters: Edges to be added per step m, Message propagation probability p and maximum distance a message can propagate to h_{max}. The model consists of two phases. The first phase is the growth phase, which is the same process as in the preferential attachment model with m edges added per step. Once the graph has reached the appropriate size via preferential attachment, for each vertex u we begin propagating a message. We then select m vertices which received that message e.g. $\{v_1, v_2, \ldots, v_m\}$ and direct edges $\{(v_1, u), \ldots, (v_m, u)\}$ back to message source u. We note that in the end of the process the graph will have $2mn$ edges.

The message propagation phase, in more detail, can be seen below:

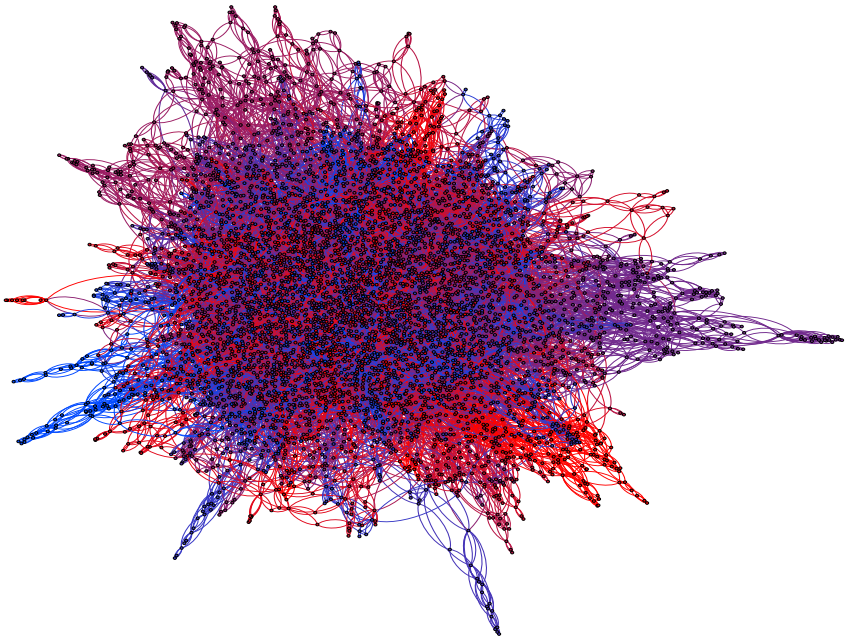


Figure 2.24: Preferential Message Propagation. $n = 6000$, $m = 1$,$p = 0.6$,$h_{max} = 8$

1. Starting from vertex u we add $< u, 0 >$ to a queue Q where $h = 0$ represents the current number of hops h.

2. While the queue Q is not empty do the following:

   2.1 Remove $< q, h >$ from Q and add q to a list L.

   2.2 For each vertex w in the neighbourhood of q in the out-direction ($N^+(q)$), if $h + 1 < h_{max}$ we add $< w, h + 1 >$ to Q with probability p.

3. We select m random vertices from L as edge endpoints for the densify process.

We note that during first step of the message propagation (or densify) procedure we expect pm vertices to retransmit the message. This is because vertex u has out-degree m. If $pm \geq 1$ then with high probability the process will terminate when $h = h_{max}$. Since the base of the process is a preferential attachment graph, which is known to be a good expander (see Section 2.1.6), we employ two tactics to ensure that the sizes of L and Q remain manageable. The first is to never add the same vertex to Q or L more

than once, which would mean that it will terminate after at most $O(n^2)$ steps (where n is the number of vertices in the graph). The second is to have an upper bound on the maximum hops h_{max} for which we will add items in the queue. In practice both these methods are used to optimize both for time and space performance In the example we present in Figure 2.23 the parameters used were: $p = 0.2$, $h_{max} = 8$ and $m = 3$. This choice of parameters was made to ensure the product $pm < 1$ and h_{max} is set to be equal to the expected diameter the graph would reach.

The above process experimentally generates power-law degree distributions on both the out-degrees and in-degrees as seen below in Figure 2.23. Additionally a community structure is created with modularity Q ranging between 0.25 and 0.75. Specifically the generated graph seen in Figure 2.24 the modularity $Q \approx 0.7$. It is empirically observed that m and p affect Q, but we have not yet determined how and why this occurs. However intuitively we assume that the aspect of the model which generates the community structure is the densification and we would expect that values of p such that $pm = 1$ will result in a higher success rate of message propagations while limiting the number of hops that the message would reach, thus more local links and therefore a better community structure. The power-law coefficient of the model presented in Figure 2.23 are $c = 3.8$, $c_{out} = 3$ and $c_{in} = 6$.

2.2.10 Grow, Back-Connect, Densify

This model is based on the Preferential Attachment With Message Propagation model mentioned in Section 2.2.9. In the context of this model we call the preferential attachment step the *grow* step, while the message propagation step the *densify* step. In addition to this we have another element in the model which is called the *Back-Connect* element. The parameters of the model are a, b, c and p where $a + b + c = 1, a, b, c \geq 0$ and $0 \leq p < 1$

Starting with a complete graph of 3 vertices, we perform the following at each step:

Grow. With probability a we add a new vertex u and choose m vertices $v_1, \ldots, v_m$

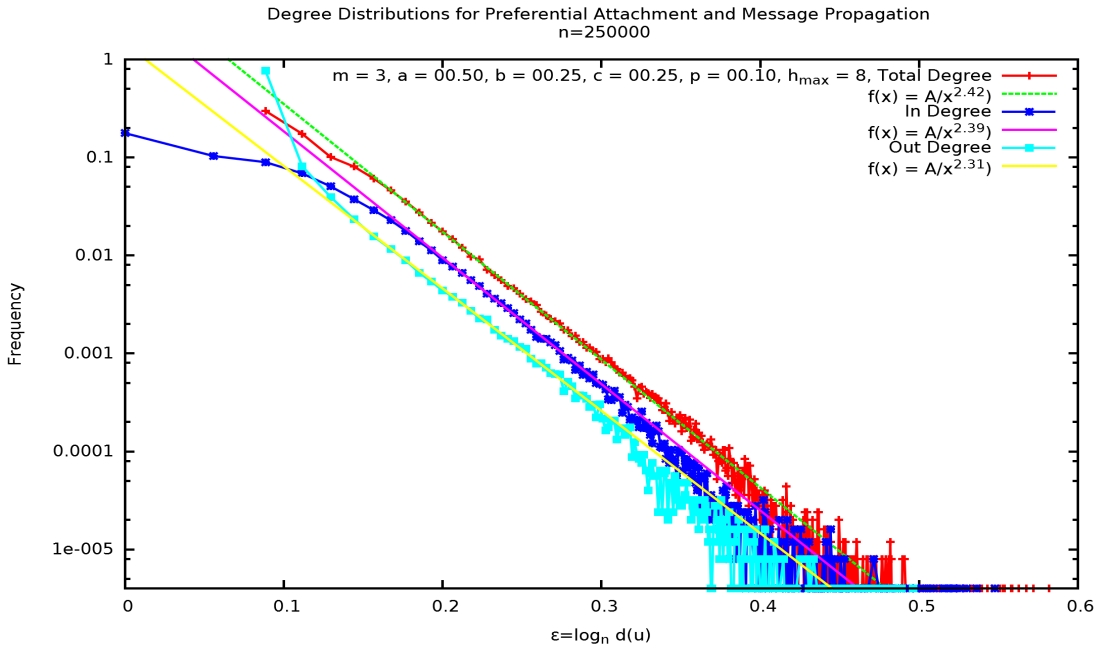


Figure 2.25: Grow-Back Connect-Densify Model. $a = 0.5, b = c = 0.25$,$p = 0.1$,$N = 200000$

preferentially and create edges (v_i, u).

Back-Connect. With probability b we choose a vertex u *uar* and a vertex v preferentially and create an edge (u, v).

Densify. With probability c we perform a message propagation similar to the one described in section 2.2.9 where p is the message transmission probability.

Given the parameters $m = 3, a = 0.5$, $b = c = 0.25$ and with $p = 0.1$ we have obtained the power-law degree distribution seen in Figure 2.25, where the resulting coefficient is approximately $c = 2.2$.

2.2.11 Partitioned Preferential Attachment

This model is essentially a combination of the well known preferential attachment model with an additional element of periodic partitioning of the graph. It is an undirected

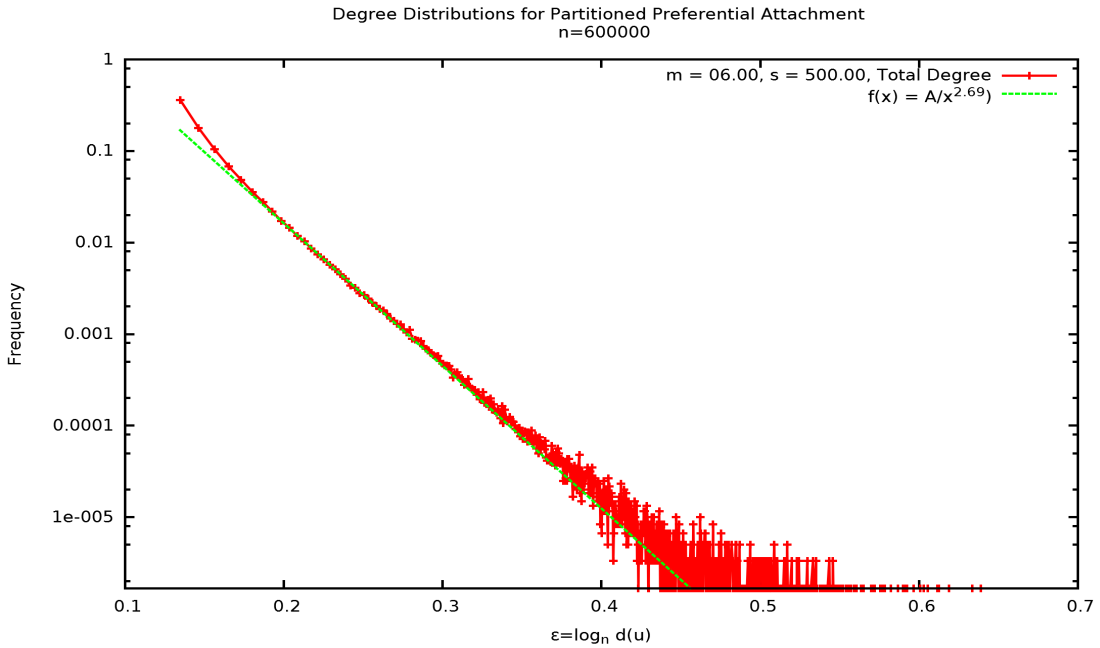


Figure 2.26: Partitioned Preferential Attachment. $n = 7,5.10^5, m = 3, s = 500$

model and it intuitively simulates the cell reproduction procedure. Given the parameters m and s, the generative procedure is as follows:

- Initially the graph is non-empty and consists of a single component.
- A new vertex u joins the graph. A component C is selected *uar* and u is added to that component.
- We select m vertices belonging to component C *preferentially*.
- We connect u to each of the m vertices selected.
- If the size of component reaches s, i.e. $|C| = s$ we create a new component C'.
- For each vertex $v \in C$ with probability $p = 0.5$ we move v to C'.

We note that during the split the edges which have endpoints belonging to different components are maintained, which maintains the connected nature of the graph. The

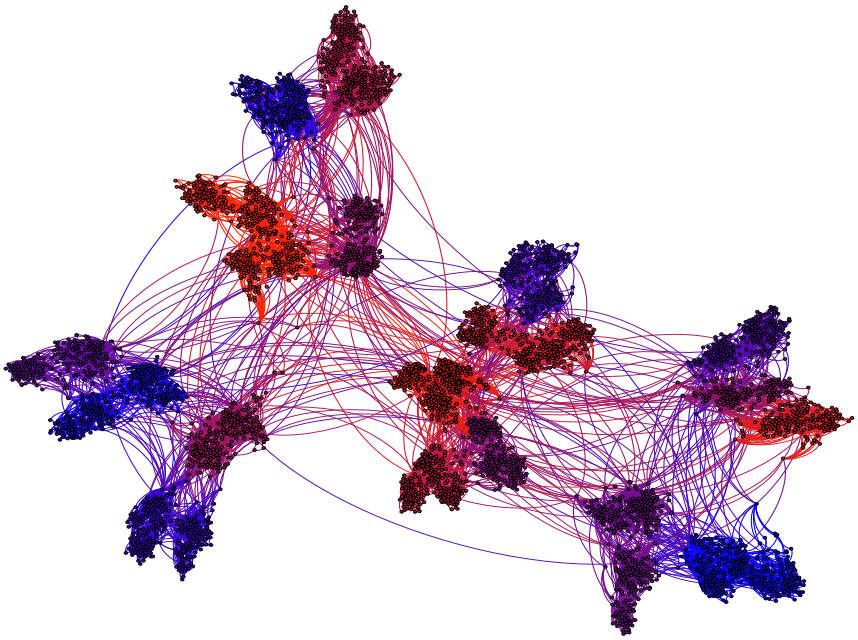


Figure 2.27: Partitioned Preferential Attachment. $n = 5000, m = 3, s = 50, Q = 0.87$

resulting degree distribution from this generative process is seen in Figure 2.26. In addition, in Figure 2.27 we can see that there is a strong community structure in graphs generated using this model.

2.2.12 Implicit Power-Law Graph Model

From an algorithmic aspect an implicit graph is a graph which is not stored in memory in advance. The structure (at least in part) can be determined when certain rules, formulae or algorithms are applied. These are given as part of the graph's definition. In this case we define a graph with the following properties:

- The graph consists of n vertices
- Each vertex is labelled u_i where $i \in [1 \dots n]$
- Given an $\alpha > 0$, each vertex u_i has an out-degree given by the formula

$$d^+(u_i) = \left\lceil \frac{A}{i^\alpha} \right\rceil \tag{2.22}$$

The intuition of this model is that, in any given OLSN users vary in activity [135]. Over time the a large number of users will not be very active, and as a result will not to create many links. A very small minority will be very active and create many links. Intuitively u_i can be viewed as the user who arrived at the network at time i. Since early users have had more time as members of the network, they would generally have been more active, while more recent users (i.e. with $i > A$) would not have had enough time to create more links within the network. In this model the parameter A acts as a limiter to the overall activity a user can have in a network.

The number of vertices of out-degree x is not explicitly given but we next prove that it its of the form of $P(d^+(u) = x) \propto x^{-(1+\frac{1}{\alpha})}$.

Theorem 2.1. *The out-degree sequence produced by formula 2.22 follows a power law with coefficient $c = 1 + \frac{1}{\alpha}$ with $a > 0$.*

Proof. We need to determine the number of natural numbers which produce the same degree, i.e., we need to know the interval for which Equation (2.22) produces the same result.

Let $i, j \in [1, \cdots, n]$ such that $d^+(u_i) = y$ and $d^+(u_j) = y + 1$ where $y \in \mathbb{N}$ and $\nexists k < i, l < j : d^+(u_k) = y, \quad d^+(u_l) = y+1$. This means that all vertices u_m with $m \in [i \dots j)$ having the same out-degree y and u_j is the first vertex in the sequence with out-degree $y + 1$.

$$\begin{aligned} y &= \frac{A}{i^\alpha} && \iff i = (\frac{A}{y})^{\frac{1}{\alpha}} \\ y+1 &= \frac{A}{j^\alpha} && \iff j = (\frac{A}{y+1})^{\frac{1}{\alpha}} \end{aligned}$$

The number of vertices with degree y, ($|D_y|$) is given by the difference between $|j - i|$. Because d^+ is a decreasing function of i, it must be that $i < j$ and therefore:

$$\begin{aligned} j - i &= (\frac{A}{y})^{\frac{1}{\alpha}} - (\frac{A}{y+1})^{\frac{1}{\alpha}} \\ &= (\frac{A}{y})^{\frac{1}{\alpha}} \left[1 - \left(\frac{1}{1+\frac{1}{y}}\right)^{\frac{1}{\alpha}}\right] \\ &= (\frac{A}{y})^{\frac{1}{\alpha}} \left[1 - (1+\frac{1}{y})^{-\frac{1}{\alpha}}\right] \end{aligned}$$

The term $(1+\frac{1}{y})^{-\frac{1}{\alpha}}$ can be approximated using a Taylor series as follows:

$$(1+\frac{1}{y})^{-\frac{1}{\alpha}} \approx (1-\frac{1}{\alpha y})$$

since $\frac{1}{y}$ is small, we omit the rest of the terms in the series without introducing a significant error. This results in:

$$\begin{aligned} j-i &\approx (\frac{A}{y})^{\frac{1}{\alpha}}\left[1-(1-\frac{1}{\alpha y})\right] \\ &= (\frac{A}{y})^{\frac{1}{\alpha}}\left(\frac{1}{\alpha y}\right) \\ &= A^{\frac{1}{\alpha}}\left(\frac{1}{\alpha y^{1+\frac{1}{\alpha}}}\right) \end{aligned}$$

Therefore the number of vertices $|D_y|$ which have a degree of y is:

$$|D_y| \approx A^{\frac{1}{\alpha}}\left(\frac{1}{\alpha y^{1+\frac{1}{\alpha}}}\right) \tag{2.23}$$

From Equation (2.23) we can conclude that $|D_y| \propto y^{-(1+\frac{1}{\alpha})}$ as $y \to \infty$ which means the function d^+ will follow a power-law in the midrange with coefficient

$$c = 1+\frac{1}{\alpha} \tag{2.24}$$

□

Since $0 < \alpha < \infty$ the power-law ranges from $1 < c < \infty$.

The only remaining issue for this model is determining the actual edges of the graph. In order to do this we follow the following procedure:

- For vertex u_i we determine the edges $\{(u_i, v_1), \ldots, (u_i, v_d)\}$ where $d = d^+(u_i)$.
- To determine the vertices $v_1, \ldots, v_d$ we use a seeded pseudo-random number generator with i as the seed.
- Using the seeded random number generator we proceeded to generate $d^+(u_i)$ consecutive random numbers. For each of these numbers j we ensure that $0 < j \leq n$.

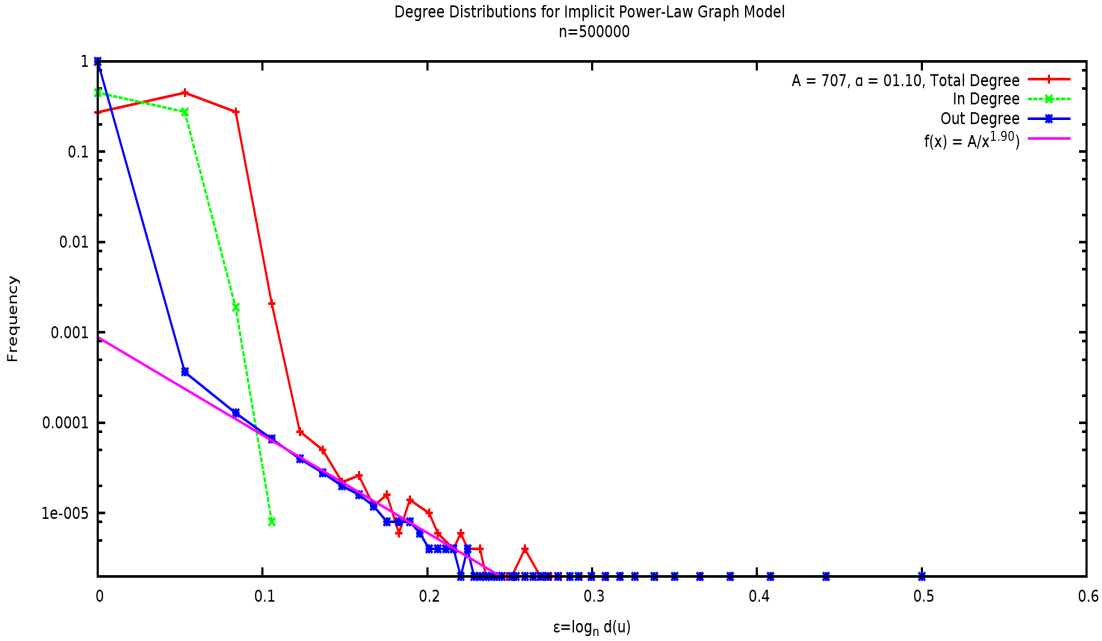


Figure 2.28: Implicit Graph Model. $\alpha = 1.25$,$N = 10000$,$c = 1.8$

- For each number j generated we add edge (u_i, u_j).

Since the pseudo-random number generator is seeded, generating a random number using the same seed always generates the same number. The above process guarantees that while the edges are effectively random, they follow a specific rule, and the same graph can be reproduced on demand.

As shown above the model generates graphs with a power-law degree distribution with coefficients which include the range $c \leq 2$ as given by Equation (2.24) if $\alpha \geq 1$. Coefficients $c \leq 2$ are hard to obtain using other generative processes.

For values of $\alpha < 1$ we must set N sufficiently large to observe power-law degree distributions. This is due to the fact that the formula at 2.22 decreases much slower. The degree distribution we obtain from this graph model is seen in Figure 2.28.

We can observe that the out-degree distribution follows a power-law, as expected by definition. The in-degree distribution approaches a log distribution but has some anomalies

which may be a direct effect of the way we determine the target vertices of the model's edges. The modularity $Q = 0.6$ which is significantly higher than expected however we currently consider this to be an artefact of the edge endpoint selection in conjunction with the fact that there are several disconnected components within the graph. It is worth noting however that the graph mainly consists of a single large component which in this case contains over 99% of the vertices of the graph while the second largest component consists of only 8 (0.08%) vertices.

2.3 Conclusion

In this section we have defined scale-free graphs and have seen some basic graph properties which describe these graphs. Specifically a scale-free graph is a graph with a power-law degree distribution, small diameter with $\text{Diam}(G) = O(\log n)$, and a scale invariance w.r.t. these properties. These kinds are the result of modeling real world networks such as: OLSNs, the WWW, autonomous systems networks, citation networks as well as biological networks such as the yeast protein-protein interaction network and many more. We have presented some properties which we used to quantify certain aspects of the network, such as e.g. Q modularity to quantify the quality of the community structure of OLSNs, as well as the clustering coefficient as an indicator of the locality of edges.

In addition we have presented some existing models for generating graphs which have some or all the properties which are observed in real world scale-free graph. Such models are generally based around the notion of growth and preferential attachment, i.e. the probability a vertex gets connected by an edge is proportional to its degree. There are many models which already exist and make use of these ideas, either directly (e.g. the Barabási-Albert model and Generalized web-graph model) or indirectly (e.g. the triangle closing model, the edge copying model etc.).

We have also presented some graph generation models which we have created. These models are:

- Random walk graph model.
- Preferential attachment and message propagation model.
- Grow,Back-Connect,Densify Model.
- Partitioned preferential attachment model.
- Implicit Power-Law Graph model.

Each of these models was created to capture additional properties of real world scale-free graphs. For example the random walk graph model was created as a generalization between the triangle closing model and preferential attachment model where by modifying a parameter of this model we can generate graphs from the former, the latter or a graph which is in between the two models. The preferential attachment and message propagation model, as well as the grow,back-connect,densify model were designed to model user behaviour patters in OLSNs such as the creation of edges based on propagation of content through these OLSNs. In addition we have created an implicit graph model which would produce a power-law with coefficient $c > 1$ while at the same time modeling user activity as a power-law. Specifically in a OLSN network, there are a majority of users with low activity who create few links, but a small minority which create many links. We prove that the consequence of such behaviour would be a degree distribution which follows a power-law and depending on the difference in activity between low-activity and high-activity users, we get a power-law slope which better corresponds to the observed slopes in OLSNs.

We have performed experiments with each model presented and have shown that in many aspects, the generated graphs share a number of common characteristics with real world scale-free graphs, such as the degree distribution, diameter and community structure.

Chapter 3

Markov Chains And Random Walks

A Markov Chain is a random process named after Andrey Markov, a Russian mathematician whose work was on these same stochastic processes. Since, in a Markov Chain the system changes randomly, it is impossible to deterministically predict the state of a Markov Chain at a future step. However, the expected future states of the system can be analysed statistically and predicted with reasonable confidence. This analysis is often referred to as Markov Chain analysis. In many applications, such as ours, it is this analysis which makes the use of Markov Chains of interest.

Markov Chains have been analysed in depth by various mathematicians and there is a rich assortment of resources available. We make extensive use of such analysis found in the book by Aldous and Fill [2]. Based on our motivation we are interested in the applications of such chains on large online networks (e.g. WWW or OLSNs) represented as graphs. A discrete-time Markov Chain whose state-space and transition probabilities can be represented as a weighted graph is often referred to as a RW on a graph. We therefore focus on analysis made specifically for RWs on graphs.

The basic framework for random walks on graphs is provided by L. Lovász [95] in his extensive survey of random walks. Additionally we build upon the work of C. Cooper *et al.* [25, 31] to provide an analysis suited for the kinds of graphs we are targeting.

Methods based on RWs are commonly used for graph searching and crawling, and such methods have been used and analysed extensively. Stutzbach *et al.* [123] compare the performance of BFS with a RW and a Metropolis Hastings Random Walk (MHRW) [61, 104] on various classes of random graphs. They use this as a basis for sampling the degree distribution of the underlying networks. The purpose of the investigation was to sample from dynamic P2P networks. In a related study, M. Gjoka *et al.* [57] made extensive use of the above methods to collect a sample of Facebook users. As SRWs are degree biased, they used a re-weighting technique to unbias the sampled degree sequence output by the random walk. This is referred to as a Re-Weighted Random Walk (RWRW) in [57]. In both the above cases it was shown the bias could be removed dynamically by using a MHRW and selecting an appropriate target distribution.

Sampling using Markov Chains, was done by A. H. Rasti *et al.* [117] for the special case of the MHRW and compared to sampling using Respondent-Driven Sampling (RDS), which can be seen as a sociological equivalent to the BFS method, in which only a subset of the neighbours of a vertex is traversed. In their work they compared the efficiency of these methods on five classes of graphs: (a) Erős-Rényi Random Graph, (b) Small World graph (Watts and Strogatz [128]), (c) Barabási-Albert Preferential Attachment Graph, (d) Hierarchical Scale-Free graph which has a structure of clusters within clusters (Barabási and Albert [6]) and (e) real online network data from the Gnutella network obtained by the authors in the context of their previous work in [122]. This examination showed the effect of graph structure and size has on the efficiency of these methods. Specifically, they used the methods of MHRW and RDS to sample for the degree distribution of the aforementioned networks. The convergence rate of the methods was measured using a Kolmogorov-Smirnov Test (K–S test) (see Section 4.3.3) and the results were presented for each method individually. It was shown that the above methods of MHRW and RDS, when applied to the Hierarchical Scale-Free had a reduced sampling accuracy, as indicated by the bad K–S test scores.

Methods such as the Re-Weighted Random Walk (RWRW) and MHRW can be used to gather unbiased samples of graphs and in particular the method of MHRW was also

used by Krishnamurthy *et al.* [74] to sample the Twitter network. They collected three samples, one using a SRW on the Twitter digraph, one using the public timeline of users who have recently posted status updates, and the third one was using a MHRW. The purpose of their study was to determine the level of symmetry between Twitter users i.e. if user u is following user v, then we say there is a symmetric edge if user v reciprocates by following user u. In addition according to the amount of symmetry for each user, they were classified according to the proportion of friends/followers they had. Users with the ratio of friends/followers approaching 1 indicate users who use Twitter as a social network, where they connect to their acquaintances and these connections are usually symmetric. Users who have a much higher number of friends than followers may indicate spammers, stalkers or simply users who friend many other users in hopes they get some of those users to reciprocate. Finally users who have a much higher number of followers than friends are broadcasters of information. These users include news companies as well as celebrities. The work of Krishnamurthy used the method of MHRW and compared the results with the results acquired by using their other methods. This was done in order to determine the level of bias the other methods had. This shows a certain level of confidence in the unbiased nature of the MHRW.

One can consider numerous sampling goals which are sometimes case specific. For example one could consider the goal of measuring cover time of a graph, that is, the expected number of steps required for a RW to visit all vertices at least once; or of determining where a RW of t steps and starting from a vertex u is most likely to terminate at(i.e. the distribution of $P_u^{(t)}$). These are some examples of possible interesting questions that arise which always depends on the problem at hand. In general, sampling with RWs is used extensively in the context of estimating structural properties of graphs. This is further examined in Chapter 5. In addition to property estimation, Markov Chains have also been used in many other areas, such as in the context of property testing. For example in the work of Czumaj *et al.* [40], RWs were used to test whether or not a graph is an expander. This was done by running multiple random walks starting from a specific vertex and stopping after a certain number of steps. If a sufficient number of

walks stopped at a specific vertex then this was an indication of a "bottleneck" in the graph. This would mean that the graph in question was not an expander.

3.1 General properties of Markov Chains

Since Markov Chains, and RWs in particular, are fundamental to our work, we first introduce some concepts and properties of Markov Chains in general. We then continue to provide properties specifically for the class of Markov Chains we will be using i.e. RWs. The concepts and properties defined here will be used in the rest of the thesis. To simplify the notation, we will mainly focus on discrete time chains, while noting that in most cases there is a direct correspondence of the same properties in continuous time chains.

A Markov Chain is a random process in a finite or countable state space, with the Markov property. Thus the conditional probability distribution for the system at the next step depends *only* on the current state of the system. We first define the notion of the *starting point*. The starting point of a Markov Chain, is the point that the chain occupies at time $t = 0$. We denote this as X_0. By extension we will denote the state occupied by the Markov chain at time t as X_t.

Based on the above notation we describe the Markov property as:

$$P(X_t = u_t \mid X_{t-1} = u_{t-1}, X_{t-2} = u_{t-2}, \cdots, X_0 = u_0) = P(X_t = u_t \mid X_{t-1} = u_{t-1}) \tag{3.1}$$

meaning that the probability to move to state u_t at time t given that at time $t-1$ we are at state u_{t-1} does not depend on the states occupied by the chain at earlier steps.

The Markov Chain M is a sequence of random variables $M = (X_0, X_1, X_2, \ldots, X_t)$ which satisfy the Markov Property (3.1). Usually, the term Markov Chain is used to mean a Markov process which has a discrete (finite or countable) state-space. Markov Chains can be either in discrete time or continuous time. Discrete time chains are often referred to as discrete RWs, and for these chains there is the notion of step, as a discrete unit. In

this case the system is assumed to occupy a certain state momentarily and the change of states is considered as an incrementation of the step. The steps are often thought of as moments in time, but they can equally well refer to physical distance or any other discrete measurement.

There are many variations of Markov Chains but we only focus on a specific one called a time-homogeneous Markov Chain. This type of Markov Chain is time independent meaning that:

$$P(X_t = u \mid X_{t-1} = v) = P(X_{t-1} = u \mid X_{t-2} = v) = \cdots = P(X_1 = u \mid X_0 = v)$$

where $P(X_t = u)$ is read as "the probability the chain occupies vertex u at time t.

A time-homogeneous Markov Chain can be described by a transition probability matrix $\boldsymbol{P}$ over the state space S. The transition probability matrix $\boldsymbol{P} \in R^{|S|\times|S|}$ is a matrix whose entries are:

$$\begin{aligned} p(u,v) = P(X_t = v \mid X_{t-1} = u) \qquad & \forall u, v \in S, p(u,v) \in [0 \ldots 1] \\ \sum_{v \in S} p(u,v) = 1 \qquad & \forall u \in S \end{aligned} \tag{3.2}$$

This can be read as "the probability to get from vertex u to vertex v in the next transition".

Extending on Equation (3.2) we note that the probability of reaching state v starting from state u after t transitions have been made, is given by the (u, v) entry of the matrix $\boldsymbol{P}^t$ and is written as:

$$\begin{aligned} p^{(t)}(u,v) = P(X_t = v \mid X_0 = u) \qquad & \forall u, v \in S, p^{(t)}(u,v) \in [0 \ldots 1] \\ \sum_{v \in S} p^{(t)}(u,v) = 1 \qquad & \forall u \in S \end{aligned} \tag{3.3}$$

Because the chains we are dealing with are time-homogeneous $P(X_t = v \mid X_0 = u)$ can be rewritten as $P(X_{t+r} = v \mid X_r = u)$. Thus we can consider each transition of a chain

to be the beginning of a new chain due to the time-homogeneous nature of this class of Markov Chains.

In summary, given a state space S, a time-homogeneous a Markov Chain is a random process in which the Markov property holds (see Equation (3.1)). The transition probability matrix $\boldsymbol{P}$ is defined in Equation (3.2).

We note some facts about random walks, which can be found either in J.R. Norris [113], Aldous and Fill Chapters 2-3 [2] or Lovasz [95]. In our notations we note that we will use T_u to denote the distribution of times at which the Markov Chain occupies vertex u. We denote as $\mathbf{E}_u T_v$ as the expected times a Markov Chain, starting at vertex u occupies vertex v. General properties which a Markov Chain may or may not have are:

Accessible and Communicating States. Given a state space S of a Markov Chain, and states $u, v \in S$ then state v is called *accessible* from state u (written as $u \to v$) if there exists a time t such that:

$$P(X_t = v \mid X_0 = u) = p^{(t)}(u, v) > 0 \tag{3.4}$$

Thus there is a non-zero probability that the chain starting from state u will reach state v after a finite number of steps t (see Equation (3.3)).

In a similar fashion, state u is said to *communicate* with state v (written as $u \leftrightarrow v$) if $u \to v$ and $v \to u$.

Communicating Class. A closed communicating class C of a state space S of a Markov Chain is the subset of S such that $\forall u, v \in C \quad u \leftrightarrow v$ and $\forall w \in S \backslash C, \quad u \not\leftrightarrow w$. That is all states in C communicate with each other and do not communicate to any state in $S \setminus C$.

Reducibility. In an irreducible Markov Chain means that all states can be reached after a certain number of steps from every possible start state of the chain, i.e. $P(X_t = v \mid X_0 = u) = p_{uv}^{(t)} > 0 \quad \forall u, v \in S$ and some positive $t \in \mathbb{N}$. If the chain

is not irreducible then it is reducible meaning that there is an initial state v from which we cannot reach the entire state space at a given step.

Alternatively assume a state-space S which can be split into n disjoint sets $C_1, \ldots, C_n$ such that $C_i \subseteq S$, with $1 \leq i \leq n$ and C_i is a communicating class. We say a Markov Chain is reducible if $n > 1$ and irreducible if $n = 1$ i.e. there is a single communicating class $C_1 = S$.

Periodicity. A state u has a period k if any return to state u must occur in times which are a multiple of k, i.e., $k = \gcd\{t : P(X_t = u \mid X_0 = v) > 0\}$. Thus k is the GCD of all values of t for which the probability to be at state u is greater than 0, if $k > 1$ then the chain is periodic otherwise it's aperiodic.

Recurrence and transience. Assume $P(X_t = u)$ is the probability the Markov Chain occupies state u at time t. Let $\mathbf{T} = \{t \mid P(X_t = u) > 0\}$. The state u is called positive recurrent if the size of the set $|\mathbf{T}| = \infty$ and transient if the set $\mathbf{T}$ is finite. There is always a positive probability that the chain will return to u if u is positive recurrent. Otherwise if u is transient, means that after some point the chain never returns to u. It can be shown that ([113] Theorem 1.5.3):

1. If $P_u(T_u < \infty) = 1$ then u is recurrent and $\sum_{t=0}^{\infty} p^{(t)}(u, u) = \infty$
2. If $P_u(T_u < \infty) < 1$ then u is transient and $\sum_{t=0}^{\infty} p^{(t)}(u, u) < \infty$

The following two theorems are from Norris [113] and hold for all communicating classes (connected components) of a graph G.

Theorem 3.1 ([113] Theorem 1.5.4)**.** *Let C be a communicating class. Then either all states in C are recurrent or all are transient*

Theorem 3.2 ([113] Theorem 1.5.5)**.** *Every recurrent class is closed.*

A Markov Chain is said to be positive recurrent if all its states are positive recurrent.

Ergodicity. Ergodicity is an additional important property of Markov Chains. A state $u \in S$ is said to be ergodic if it is aperiodic, i.e. if $k = 1$ in the definition of periodicity. A Markov Chain is said to be ergodic if it is aperiodic and irreducible.

First Hitting Time. Given an ergodic Markov Chain starting from a vertex u, the first hitting time of vertex v, $H(u, v)$ is defined as:

$$H(u, v) = \min\{t : X_0 = u, X_t = v\} \tag{3.5}$$

For a walk starting in state v, define the first return time to state v as

$$H(v, v) = T_v^+ = \min\{t : X_0 = v, X_t = v, t > 0\}. \tag{3.6}$$

Absorbing states A state u is called an absorbing state if $P(X_t = u \mid X_{t-1} = u) = 1$ and $P(X_t = v \mid X_{t-1} = u) = 0 \quad \forall v \neq u$. The chain cannot "escape" from an absorbing state.

A Markov Chain with absorbing states is called an absorbing chain, if every non-absorbing state can reach an absorbing state in a finite number of steps. Additionally all non-absorbing states in an absorbing Markov Chain are transient

In addition to Markov Chains which obey the Markov property defined in Equation (3.1), there are some relaxations which are allowable in some cases. For example for some classes of Markov Chains the $P(X_t = u \mid X_{t-1} = v, \ldots X_0 = w)$ does not depend on just the previously occupied state, but rather, the m previously occupied states, for a finite m. These chains are are called higher order Markov Chains, (or order m Markov Chains). While a transition probability exists for such chains, it depends on the m previous steps: $P(X_t = x_t \mid X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \cdots, X_{t-m} = x_{t-m})$. Therefore these order m Markov Chains can be expressed by a Markov Chain with no memory and with a state space $S_m = S^m$ meaning that each state is an m-dimensional vector over the original state space. This means that the analysis of Markov Chains with no memory also holds

for higher order Markov Chains. However, basic properties of a Markov Chain, such as recurrence etc. which may hold over state space S, may not necessarily hold for state space S^m.

We now define the notion of *stationary distribution*. The stationary distribution is a fundamental property of time-homogeneous Markov Chains. This distribution has various ways to be defined and we will present them here:

Definition 3.1 (Stationary Distribution)**.** *Given a Markov Chain, the stationary distribution* $\boldsymbol{\pi}$ *is the probability distribution which satisfies the following balance equation:*

$$\pi_v = \sum_u \pi_u p(u, v) \tag{3.7}$$

We note that the stationary distribution is not necessarily unique. We will now provide the definition of the limiting stationary distribution based on matrix $\boldsymbol{P}$.

Definition 3.2 (Limiting Stationary Distribution)**.** *The limiting distribution* Π *of Markov Chains is said to exist if the following limit exists:*

$$\lim_{t\to\infty} \boldsymbol{P}^t = \Pi, \tag{3.8}$$

If the limit exists then the limiting stationary distribution of the Markov Chain is Π.

The stationary distribution of a state π_u can be understood as "The probability of a Markov Chain to be at a specific state at a specific time after a large number of steps". If the stationary distribution exists, then it is independent of the initial state meaning that in a sense the chain "forgets" where it started from and the probability it currently occupies a state u only depends on π_u.

The stationary distribution may not necessarily be unique for a Markov Chain. However if the chain is Ergodic then it defines a unique stationary distribution. In this case the Ergodic theorem holds. The Ergodic theorem can be viewed as the analogue of the strong law of large numbers but applied to Markov Chains, and is defined as follows:

Theorem 3.3 (The Ergodic theorem: [2] Chapter 2 Theorem 1)**.** *Let* $N_u(t)$ *be the number of visits to state* u *during times* $0, 1, \ldots, t-1$*. Then the following holds:*

$$t^{-1} N_u(t) \overset{a.s.}{\to} \pi_u, \quad \text{as } t \to \infty \tag{3.9}$$

where π_u *is the stationary distribution of* u *(see Definition 3.1).*

This means that $t^{-1}N_u(t)$ converges almost surely to π_u, i.e., $P(\lim_{t\to\infty} t^{-1}N_u(t) = \pi_u) = 1$.

We next outline a simple explanation on why periodic chains do not necessarily have a uniquely defined stationary distribution. We note that in the case that the chain has a periodic state u with period k then:

$$\lim_{t\to\infty} p^{(t)}(u,u) = \begin{cases} c & t \bmod k = 0 \\ 0 & \text{otherwise} \end{cases}$$

where $0 < c \leq 1$. In other words, the limit does not exist in those cases and by extension the limiting stationary distribution matrix Π does not exist. There may still be a distribution which satisfies Equation (3.7) in this case, i.e., the stationary distribution may still exist. In Section 3.4 we discuss a way to eliminate the periodicity of a chain and ensure the existence of the limit. If we assume an aperiodic chain with all states being positive recurrent then the stationary distribution exists and is unique for each communicating class C_i of the chain. The matrix Π is an $|S| \times |S|$ matrix but is of rank k where k is the number of communicating classes in the chain. If the chain is irreducible then $k = 1$ and Π can be simply referred to as the unique vector π which is the row vector to which all rows of matrix Π converge to. From here on we will only be dealing with ergodic Markov Chains which define a unique stationary distribution π and we will refer to the distribution of each state u in the chain as π_u.

For a chain which starts at v, the expected value of the first return time of state v is related to the stationary distribution of v as follows:

$$\mathbf{E}_v T_v^+ = \frac{1}{\pi_v} \tag{3.10}$$

We will refer to $\mathbf{E}_v T_v^+$ as $\mathbf{E}T_v^+$.

3.1.1 Reversibility

In our work we will mostly deal with *reversible* time-homogeneous discrete time Markov Chains over a discrete state space. Specifically in our work we will be dealing with

Random Walks (RWs) on undirected graphs, which are time-homogeneous reversible Markov Chains. The name RW comes from the fact that they can be represented as a particle which, at each discrete time takes a random step among states based on the transition rule which satisfies the transition probability matrix $\boldsymbol{P}$.

This section introduces the definition of *reversibility* as presented in [2, 113].

Definition 3.3 (Reversibility)**.** *A Markov Chain which defines a unique stationary distribution $\boldsymbol{\pi}$ is called reversible if following detailed balance equation holds:*

$$\pi_u p(u,v) = \pi_v p_(v,u) \quad \forall u,v \in S \tag{3.11}$$

The detailed balance equation shown in (3.11) is a special case of Equation (3.7).

The name *reversible* comes from the fact that, if we assume a chain starting from stationarity π at state X_0 the following holds for all $t($ [2]):

$$(X_0, X_1, \ldots, X_t) \stackrel{d}{=} (X_t, X_{t-1}, \ldots, X_0)$$

meaning that the chain $(X_0, X_1, \ldots, X_n)$ has the same distribution as its time-reversal $(X_t, X_{t-1}, \ldots, X_0)$.

3.1.2 Fundamental matrix Z

The fundamental matrix **Z** is an important tool used in Markov Chain analysis. It is defined as:

$$Z(u,v) = \sum_{t=0}^{\infty} \left(p^{(t)}(u,v) - \pi_v \right) \tag{3.12}$$

where $p^{(t)}(u,v)$ is probability a walk starting from u visits v at step t.

The following useful lemma can be found in [2] (Chapter 2. Lemma 11, 12)

Lemma 3.1.

$$\pi_u \mathbf{E}_\pi T_u = Z_{uu}$$
$$\pi_v \mathbf{E}_u T_v = Z_{vv} - Z_{uv}$$

3.1.3 Mixing times–Relaxation times

As we noted above, an ergodic Markov process has a unique stationary distribution π. This stationary distribution can be determined by Equation (3.8) or Equation (3.28) for random walks on graphs. The rate in which $P^k \to \Pi$ is discussed in Aldous and Fill [2] Chapter 3 and Lovasz [95].

It is worth noting that since P is a probability matrix and assuming the set of the absolute values of the eigenvalues of this matrix is $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{|S|}\}$ sorted by value, then the largest eigenvalue is $\lambda_1 = 1$ and the second largest eigenvalue is λ_2. For discrete ergodic chains, the gap $1 - \lambda_2$ has the interpretation of "asymptotic rate of convergence to the stationary distribution".

For a walk starting at vertex u, the following bound holds ([95]):

$$|P^{(t)}(u, v) - \pi_v| \leq \sqrt{\frac{d(v)}{d(u)}}\lambda_2^t \tag{3.13}$$

The *ϵ-mixing time* $T(\epsilon)$ of a walk is the number of steps required to converge to within a distance ϵ from $\boldsymbol{\pi}$, i.e.,

$$|P^{(T(\epsilon))}(u, v) - \pi_v| \leq \epsilon$$

Since the infinite geometric series $\sum_{t=1}^{\infty} \lambda_2^t = \frac{1}{1-\lambda_2}$ the mixing time is often defined in terms of the *relaxation time* τ_2 ([2, 73]):

$$\tau_2 = \frac{1}{1 - \lambda_2} \tag{3.14}$$

We note that for continuous chains $\tau_2 = \frac{1}{\lambda_2}$. The relaxation time τ_2 has a meaning of an upper bound of convergence to stationarity.

In many cases however the quantity λ_2 is hard to obtain. For stationary Markov Chains on graphs, the relaxation time can be bounded in terms of conductance, which is a quantity easier to determine or estimate. This is done based on the Cheeger inequality [20]:

$$\frac{\Phi^2}{2} \leq 1 - \lambda_2 \leq 2\Phi \tag{3.15}$$

In Equation (3.15) $\Phi = \Phi_G$ is the conductance of the graph, first introduced in Chapter 2 Equation (2.4). We will choose to use the simpler notation:

$$\phi(S) = \frac{e(S : \overline{S})}{\min(d(S), d(\overline{S}))}$$

where $e(S : \overline{S})$ denotes the edges starting from S and ending outside of S and $d(S)$ denotes the degree of S. We remind that Φ_G, the conductance of the graph, is the minimum value of $\phi(S)$ over all possible sets $S \subseteq V$ with $|S| \leq \frac{|V|}{2}$.

3.1.4 Stopping times, hitting times and return times

We focus only on time-homogeneous Markov Chains which are ergodic, and therefore define a unique stationary distribution. Since our work focuses mainly on graphs we refer to these chains as *Random Walks*. We only consider reversible chains, however most of the properties presented in this section hold for both reversible and irreversible chains unless otherwise specified.

We now proceed to present some related definitions which are derived from the Ergodic principle which is $\frac{N_t(u)}{t} \overset{a.s.}{\to} \pi_u$(see Theorem 3.3). For a walk starting from a state u, let S be a random stopping time such that $X_S = u$ (and $E_u\ S < \infty$), e.g. each time $X_t = u$ we decide with some probability to stop the chain; if the chain stops at step t then $S = t$. Then the following hold:

$$\mathbf{E}_u(\text{number of visits to } v \text{ before time } S) = \pi_v \mathbf{E}_u\ S. \tag{3.16}$$

In particular, if $S = T_u^+$ then

$$\mathbf{E}_u(\text{number of visits to } v \text{ before time } T_u^+) = \frac{\pi_v}{\pi_u}.$$

We remind that $\mathbf{E}_u$ is used to denote the expectation for chains starting at state u.

3.1.5 Hitting Time From Stationarity

The expected hitting time of vertex v from stationarity is given by the following([2]):

$$\mathbf{E}_\pi T_v = \sum_{u \in V} \pi_u \mathbf{E}_u T_v,$$

where $\mathbf{E}_u T_v$ is the expected time to hit v starting from u. Conveniently $\mathbf{E}_\pi T_v$ can be expressed as

$$\mathbf{E}_\pi T_v = \frac{Z_{vv}}{\pi_v} \tag{3.17}$$

where Z_{vv} is the (v, v) value of the fundamental matrix Z defined in Section 3.1.2.

The variance of the first return time T_v^+ is given by (see e.g. [2])

$$\mathbf{Var}\ T_v^+ = \frac{2\mathbf{E}_\pi T_v + 1}{\pi_v} - \frac{1}{\pi_v^2}. \tag{3.18}$$

combining this with Equation (3.17) we get:

$$\begin{aligned} \mathbf{Var} T_u^+ &= \frac{2\mathbf{E}_\pi T_u + 1}{\pi_u} - \frac{1}{\pi_u^2} \\ &= \frac{2\frac{Z_{uu}}{\pi_u} + 1}{\pi_u} - \frac{1}{\pi_u^2} \\ &= \frac{\pi_u(2\frac{Z_{uu}}{\pi_u} + 1)}{\pi_u^2} - \frac{1}{\pi_u^2} \\ &= \frac{2Z_{uu} + \pi_u - 1}{\pi_u^2} \end{aligned} \tag{3.19}$$

$$\tag{3.20}$$

We can combine the above with Equation (3.13) in page 100 ([95])

$$|p^{(t)}(v, v) - \pi_v| \leq \lambda_2^t$$

so we can obtain

$$1 - \pi_v \leq Z_{vv} \leq \frac{1}{1 - \lambda_2}. \tag{3.21}$$

Setting $C(v) = C = 2Z_{vv} + \pi_v - 1$ we get:

$$\mathbf{Var}\ T_v^+ = \frac{C}{\pi_v^2} = C(\mathbf{E}T_v^+)^2 \tag{3.22}$$

where

$$1 - \pi_v \leq C \leq 2/(1 - \lambda_2).$$

3.2 Simple Random Walk (SRW)

As mentioned in Section 3.1.4 our work focuses mainly on graphs. In this section we define some special cases of Markov Chains whose state space S can be viewed as the vertex set V of a graph $G = \{V, E\}$. Commonly the edge set E of the graph corresponds to permitted transitions i.e. transitions which have a non-zero probability of occurring. However this is not always the case and we will see many examples of random walks on graphs which also permit transitions between vertices which are not connected by an edge. If we were to model the Markov Chain precisely as a graph this would not be the case since we would need to insert these edges in the edge set. Since the graphs we use usually correspond to an underlying online network, such an addition would not be permitted.

We also note that from this point forward we will only be dealing with *ergodic* and *reversible* chains, unless otherwise stated. However many of the properties which will be presented may also apply to other chains. A SRW is a Markov Chain on a graph, in which each transition is done towards a randomly selected adjacent vertex of the current state. In general we will mainly focus on undirected graphs and this is done for various reasons. The main reason is that in the case of the large online networks we will be targeting, some of them are undirected by default (e.g. Facebook). The ones that are not undirected usually allow queries in either direction allowing us to ignore direction and simply view them as undirected (e.g. Twitter). This would allow us to take advantage of reversibility properties, such as the detailed balance equation.

Let $G = \{V, E\}$ be a graph. A SRW W_u, $u \in V$ on graph G is a Markov Chain $X_0 = u, X_1, \ldots, X_t, \ldots$ on the vertices V associated to a particle that moves from vertex to vertex according to a transition rule. Specifically if the particle is at vertex u, then the transition rule of a SRW is to select a vertex $v \in N(u)$ uniformly from the set $N(u)$, where $N(u)$ is the set of the neighbours of u.

Definition 3.4. *Assume a SRW starting at vertex u, the transition probabilities which*

would make up the transition probability matrix $\boldsymbol{P}$ *(see Equation* (3.2)*) are:*

$$p(u,v) = \begin{cases} \frac{1}{d(u)}, & \exists\{u,v\} \in E \\ 0, & otherwise \end{cases} \tag{3.23}$$

where $d(u)$ *is the degree of vertex* u*.*

We note how the quantity $f(u) = \frac{d(u)}{2m}$ satisfies the balance equation (3.7) due to the fact that:

$$\sum_{v \in V} p(u,v) f(v) = \sum_{v \in N(u)} \frac{1}{d(v)} \frac{d(v)}{2m} = f(u) \tag{3.24}$$

$$\sum_{u \in V} f(u) = \frac{1}{2m} \sum_{u \in V} d(u) = 1$$

where $m = |E|$, the total number of edges in G.

This means that $f(u) = \pi$ is a stationary distribution for the SRW. Additionally if the underlying graph is connected then the Markov Chain is irreducible and therefore, assuming the chain is aperiodic, π is unique. We can ensure that even on bipartite connected graphs the chain is aperiodic and therefore the stationary distribution is unique. This is discussed in Section 3.4.

In the cases that the graph is not connected and consists of k connected components then we can decompose the vertex set into k disjoint sets $\{G_1, G_2, \ldots, G_k\}$ with $G_i = \{V_i \subset V, E_i \subseteq E\}$ such that each graph G_i is a connected component of G. In this case V_i is a communicating class for the Markov Chain.

Assuming vertex $u \in V_i$ and the function $g(u)$

$$g(u) = \frac{d(u)}{2k|E_i|}$$

we can observe that:

$$\sum_{v \in V} p(u,v) g(v) = \sum_{v \in N(u)} \frac{1}{d(v)} \frac{d(v)}{2k|E_i|} = g(u) \tag{3.25}$$

$$\sum_{u \in V} g(u) = \sum_{i=0}^{k} \sum_{u \in V_i} \frac{d(u)}{2k|E_i|} = \sum_{i=1}^{k} \frac{1}{k} = 1$$

We observe that both $f(u)$ and $g(u)$ satisfy the balance equation and therefore are both stationary distributions which means the only case where the stationary distribution is unique is when $k = 1$ and $G_1 = G$.

By definition of recurrence, a state u is positive recurrent if $\sum_{t=0}^{\infty} p^{(t)}(u,u) = \infty$. Assuming an irreducible Markov Chain with stationary distribution given by Equation (3.25), and a finite relaxation time τ_2. Let $t = k\tau_2$ where $k \in \mathbb{N}$. From Equation (3.13) page 100:

$$|p^{(k\tau_2)}(u,v) - \pi_v| \leq \sqrt{\frac{d(u)}{d(v)}}\lambda_2^{\frac{k}{1-\lambda_2}}$$

if $u = v$ then:

$$|p^{(k\tau_2)}(v,v) - \pi_v| \leq \lambda_2^{\frac{k}{1-\lambda_2}}$$

since $a^b = \exp(b \log a)$ we have:

$$|p^{(k\tau_2)}(v,v) - \pi_v| \leq \exp\left(\frac{k}{1-\lambda_2}\log\lambda_2\right)$$

$\lambda_2 < e$ therefore $\log\lambda_2 < 0$ and so as $k \to \infty$ we have:

$$|p^{(k\tau_2)}(v,v) - \pi_v| \leq \exp\left(\frac{k}{1-\lambda_2}\log\lambda_2\right) \sim e^{-ck}$$

where $0 < c < 1$. Therefore $p^{(k\tau_2)} \approx \pi_u$ for $k \in \mathbb{N}$. We note that k needs to be large enough to make the error smaller than π_v, i.e., when $e^{-k} \ll \pi_v$.

Since the chain is irreducible, then $d(u) > 0$. This means that:

$$\sum_{t=0}^{\infty} p^{(t)}(u,u) \geq \sum_{k=1}^{\infty} p^{(k\tau_2)}(u,u) \approx \sum_{k=1}^{\infty} \frac{d(u)}{2m} = \infty$$

which means state u is positive recurrent. This assumes a relaxation time $\tau_2 < \infty$ which means that the chain is aperiodic. If the chain is periodic then $\lambda_2 = \lambda_1 = 1$ and $\tau_2 \to \infty$.

We note that a SRW is biased towards high degree vertices meaning that high degree vertices are sampled more often, compared to a uniform sample. This makes random walks unsuitable to obtain uniform vertex samples. However in Chapter 4 we describe ways to correct this bias.

3.3 Weighted Random Walk (WRW)

The idea of *WRWs* comes as a natural next step from the SRW. There are cases where the edges of a graph are weighted, usually denoting edge significance. This means that a SRW is not sufficient to represent the transitions in this specific model. The transition probability of the walk should take these weights into account.

Definition 3.5. *Given a graph $G = \{V, E\}$ and a function $w(u, v) > 0 \quad \forall\{u, v\} \in E$, the WRW on graph G is defined by the following transition probability matrix:*

$$p(u,v) = \begin{cases} \frac{w(u,v)}{\sum_{k \in N(u)} w(u,k)}, & \exists\{u,v\} \in E \\ 0, & otherwise \end{cases} \tag{3.26}$$

where $w(u, v)$ denotes the weight of edge $\{u, v\}$.

The function w can be interpreted as a weight function where the edge $\{u, v\}$ has a weight $w(u, v)$. In short the WRW is a random walk which differs from the SRW w.r.t. the transition rule. While in the SRW the transition rule is a uniform neighbour selection, in the WRW the neighbour selection is done proportionally to the weight of the associated edge.

The quantity $w(u) = \sum_{k \in N(u)} w(u, k)$ is often referred to as the *weight of vertex u* and is a basic vertex property in the context of random walks on graphs. The SRW is a special case of the WRW where $w(u, v) = 1 \quad \forall\{u, v\} \in E$ which results in $w(u) = d(u)$.

An additional fundamental property of WRWs is the *overall graph weight* which is given by Equation (3.27).

$$w_G = \sum_{u \in V} w(u) = 2 \sum_{\{u,v\} \in E} w(u, v) \tag{3.27}$$

The stationary distribution of vertex u in this chain is:

$$\pi_u = \frac{w(u)}{w_G} \tag{3.28}$$

In the case that $w(u) = d(u)$ and $w_G = 2m$ Equation (3.26) becomes equivalent to Equation (3.23). For the same reasons as for the SRW, a WRW on a connected finite graph with a finite relaxation time defines a unique stationary distribution π which is given by Equation (3.28).

3.3.1 Electrical network paradigm

In the case of WRWs, the weight $w(e)$ of an edge e has the meaning of conductance in electrical networks, and the resistance $r(e)$ of e is given by $r(e) = 1/w(e)$. Under this paradigm we can use additional properties of electrical networks such as e.g. the effective resistance between two vertices, denoted as R_{eff}.

The commute time $K(u, v)$ between vertices u and v, is the expected number of steps taken to travel from u to v and back to u. The commute time for a weighted walk is given by

$$K(u, v) = w(G)R_{eff}(u, v). \tag{3.29}$$

Here $w(G)$ is given by Equation (3.27), and $R_{eff}(u, v)$ is the effective resistance between u and v, when G is taken as an electrical network with edge e having resistance $r(e)$. For our work we do not need to calculate $R_{eff}(u, v)$ very precisely, but rather note that if uPv is any path between u and v then

$$R_{eff}(u, v) \leq \sum_{e \in uPv} r(e).$$

For $u \in V$, and a subset of vertices $S \subseteq V$, let $C_u(S)$ be the expected time taken for W_u to visit every vertex of G. The *cover time* C_S of S is defined as $C_S = \max_{u \in V} C_u(S)$. We define a walk as *seeded* if it starts in S. The *seeded cover time* ${C_S}^*$ of S as ${C_S}^* = \max_{u \in S} C_u(S)$. For a random walk starting in a set S, the cover time of S satisfies the following Matthews bound

$$C_S^* \leq \max_{u,v \in S} H(u, v) \log |S|. \tag{3.30}$$

For $u \neq v$, the variable $H(u, v)$ is the expected time to reach v starting from u (the hitting time). The commute time $K(u, v)$ is given by $K(u, v) = H(u, v) + H(v, u)$, so $K(u, v) > H(u, v)$.

3.4 Lazy Random Walk (LRW)

The Lazy Random Walk (LRW) is another special case of a Markov Chain. It is a special random walk in which which there is a guarantee that $P(X_t = u \mid X_{t-1} = u) > 0 \quad \forall u \in V$. This means that there is always a probability that the walk will chose to stay at a specific vertex rather than moving to a different one.

We will describe a two LRWs, each with a different transition probability matrix.

The simplest LRW is the random walk with the following transition probability matrix:

$$p(u,v) = \begin{cases} \frac{w(u,v)}{2\sum_{k \in N(u)} w(u,k)}, & \exists \{u,v\} \in E \\ \frac{1}{2}, & u = v \\ 0, & \text{otherwise} \end{cases} \tag{3.31}$$

Based on Equation (3.31) one can relate this walk with the general WRW but with the addition of a loop with weight $\frac{1}{2}w(u)$ and by also halving the weights of all other transitions. This results in a stationary distribution which is identical to the one seen in Equation (3.28). There is however a significant advantage in using this kind of walk and that comes from the fact that $P(X_t = u \mid X_{t-1} = u) \neq 0$. This means that $\gcd\{t : P(X_t = u \mid X_0 = v) > 0\} = 1$ as needed for an aperiodic irreducible chain. This is based on the idea that if $t_1 \in \{t : P(X_t = u \mid X_0 = v) > 0\}$ then $t_1 + 1 \in \{t : P(X_t = u \mid X_0 = v) > 0\}$ and the fact that consecutive integers are co-prime.

If a SRW or WRW on a connected undirected graph is aperiodic the stationary distribution is given by Equation (3.28). As we have seen in Equation (3.8) the stationary distribution Π only exists if $\lim_{t \to \infty} P^t$ exists, however this limit does not exist in the case of periodic chains. Using a LRW we can ensure that the Markov Chain on any graph would not be periodic, even if the underlying graph is k-partite. We will therefore assume that this sort of walk is always used to ensure the existence of a stationary distribution, (even if that stationary distribution is not unique in the underlying walk) and thus the walk is irreducible and aperiodic.

An alternative form of a LRW is described by the following transition probability matrix:

$$p(u,v) = \begin{cases} \frac{1}{n}, & \exists\{u,v\} \in E \\ \frac{n-d(u)}{n}, & u = v \\ 0, & \text{otherwise} \end{cases} \tag{3.32}$$

where $n = |V|$

In practice, the walk in Equation (3.32) can be described as follows. It is the walk in which, at a specific step we select a vertex v uniformly at random from V. A transition there is then performed if there is an edge between the current vertex u and the selected vertex v. We can view this LRW as a WRW where each vertex u has a loop with weight $\frac{n-d(u)}{n}$ and each other edge has weight $\frac{1}{n}$. This results in each vertex having weight $w(u) = 1 \quad \forall u \in V$. The stationary distribution for each vertex of this walk is $\pi_u = \frac{1}{n}$ which is the uniform distribution. This property makes it useful walk to obtain uniform samples from a graph.

3.5 Metropolis Hastings Random Walk (MHRW)

The MHRW is an other special case of a Markov Chain. It was first proposed by Metropolis *et al.* [104] and later analysed and generalized by Hastings *et al.* [61].

The idea is to use a given transition matrix $\boldsymbol{P}(u,v)$ to manufacture a different walk which samples from a desired stationary distribution $\boldsymbol{Q}$ over the set of vertices. Assume we run a SRW with transitions $p(u,v)$ from state u to state v given by the SRW transition probability matrix $\boldsymbol{P}$, then during each step the MHRW would accept v as the next vertex to visit with a probability:

$$a(u,v) = \min\left\{1, \frac{Q(v)p(u,v)}{Q(u)p(v,u)}\right\} \tag{3.33}$$

Here $Q(u), Q(v)$ are the desired sampling probabilities for vertices u and v respectively and $p(u,v) = \frac{1}{d(u)}$ if $v \in N(u)$, as given in Equation (3.23). The entries of the transition probability matrix $\boldsymbol{P}_{MHRW}$ of the MHRW are defined in Equation (3.34). Note that, if

the walk does not transition to v, it remains at u for this step.

$$p(u,v) = \begin{cases} \frac{1}{d(u)} a(u,v), & \text{if } v \in N(u), \\ 1 - \sum_{w \in N(u)} a(u,w), & v = u \\ 0, & \text{otherwise} \end{cases} \tag{3.34}$$

The MHRW is manufactured in such a way such that we can obtain any desired distribution by sampling our graph. An example of this in the context of online social networks is in Gjoka *et al.* [57] or Stutzbach *et al.* [123] where the MHRW is used to gather uniform samples from Facebook. A uniform sample requires the distribution of all vertices to be $P(u) = \frac{1}{|V|} \quad \forall u \in V$ (uniform vertex selection). The acceptance probability $a(u,v)$ was:

$$a(u,v) = \begin{cases} \min(1, \frac{d(u)}{d(v)}), & \exists \{u,v\} \in E \\ 0, & \text{otherwise} \end{cases} \tag{3.35}$$

This indicates that there are application or network specific optimizations that can be done on random walks in order to tune them to the required task. However, the downside is that for each specific distribution we select, the mixing rate of the entire chain changes in unpredictable ways.

3.6 Random Walk implementation and simulations

Since our work mainly focuses on large online networks, and as a major aspect of that work is sampling from these networks, we make extensive use of Markov Chains as a sampling tool. Due to the large size of these networks and the fact that we do not usually have much information about them, we cannot make use of matrix operations to infer e.g. stationary distribution or expected hitting times. It is often the case that we do not even know the real size of these networks or the number of edges which exist within.

To overcome all these limitations we simulate RWs as a sampling algorithm. We will discuss sampling methods and issues using RWs in Chapter 5. In this section we will discuss algorithmic complexity issues and for this purpose we present the algorithms

corresponding to the main random walks we have presented so far, the SRW and the WRW.

For the purposes of explanation we will assume that we have an arbitrary stopping criterion for all random walk algorithms in which the algorithm stops after s transitions are made. This criterion is a widely used when sampling, and is based on the practical limitations of time and space that arise from simulating RWs. We will refer the reader to Aldous and Fill [2] for properties of RWs with arbitrary stopping times such as the one seen in Equation (3.16). The RW on graphs is simulated with the algorithm presented as Algorithm 1.

Algorithm 1 RW

```
v ← start vertex
i ← 0
while i < s do
  Visit(v)
  v ← random vertex from N(v)
  i ← i + 1
end while
```

Based on this algorithm we can note that $p(u, v) = \frac{1}{d(u)}$ which means that we are indeed sampling from the transition matrix $\boldsymbol{P}$. For example if we perform a large number of such walks starting from u and record the vertex which was visited last in each case, we will observe that the $P(T_s = v) = p^{(s)}(u, v)$. If we are interested in sampling from the stationary distribution π_u we choose the stopping time s to be more than the relaxation time e.g. $s > \tau_2$. If this is true then $P^s \approx \Pi$ and $P(T_s = v) \approx \pi_v$. While in practice the properties we sample from the random walk are not so straight forward, we can see that we have a powerful tool to simulate these walks and infer relevant properties.

The complexity of Algorithm 1 is straightforward to determine. We note how the number of steps taken is s and during each step there are only 3 actions taken. In most cases, the action "Visit(v)" takes $f(v) = O(1)$ therefore the complexity of the algorithm is $O(s)$. There are cases in which $f(v)$ is not constant time, which adds to the complexity of the algorithm, and the overall complexity is generalized as $O(sf(v))$. It is clear however,

that the RW algorithm itself is of linear complexity in the number of steps s.

The WRW on graphs is in general the algorithm presented in Algorithm 2.

Algorithm 2 WRW

```
u ← start vertex
while i ¡ s do
  Visit(u)
  U ← random number between [0 · · · w(u)]
  Acc ← 0
  for all v ∈ N(u) do
    if Acc ≤ U ≤ Acc + w(u, v) then
      u ← v
      Acc ← Acc + w(u, v)
    end if
  end for
end while
```

There is additional complexity in this algorithm, arising from the fact that we need to obtain the weight of each adjacent edge of the vertex we are currently at. Each step requires a worst-case time of $O(d(u))$ (assuming $f(v) = O(1)$). There are additional optimizations that could be made to reduce the general worst-case time of each WRW step to $O(\log d(u))$, which we will discuss further in the following section.

3.6.1 Caching to improve WRW performance

When sampling using a WRW we wish to avoid the overhead which is added from determining the vertex weight at each step. To do this we make use of a cache. Firstly we assume that the vertices in $N(u)$ are indexed so $N(u) = (v_1, v_2, \ldots, v_{d(u)})$. The cache is essentially a set:

$$\{u \to \{w(u, v_1), w(u, v_1) + w(u, v_2), \ldots, \sum_{i=1}^{d(u)} w(u, v_i)\}\}$$

What is stored in this cache is a cumulative weight for each neighbour with $w(u)$ being at the last entry. If we were to generate a random number $U \in [0, \ldots, w(u)]$ then if $0 \leq U < w(u, v_1)$ we would select v_1, if $\sum_{i=1}^{k-1} w(u, v_{k-1}) \leq U < \sum_{i=1}^{k} w(u, v_k)$ we would

select vertex k etc. The check to determine the range in which U falls can be done with a binary search, which requires time $O(\log d(u))$ however this requires us to pre-process the graph in order to fill the cache with the appropriate entries. The cost of using this caching mechanism is that each vertex requires $O(1 + d(u))$ memory and therefore we would need $O(n + m)$ space to store the entire graph. Since it is not always possible to do this we make the compromise of filling in these values each time a vertex which is not already in the cache is visited. We refer to this caching mechanism as *dynamic caching*. Dynamic caching has an additional advantage when using relatively short random walks. This advantage is that not all vertices will need to be cached, but just the ones which are discovered. This makes it a much more practical method to use in the cases where $s \ll C_V$ where C_V is the number of steps required to visit all vertices of the graph i.e. the *cover time* of the graph. In practice, we have found this method to be very efficient since the most expensive vertices to get the weights for are the high degree vertices. At the same time for most of our experiments those vertices are the ones with the highest weights as well, which means they are visited more often, and therefore queried more frequently.

We have found that this caching mechanism works better in practice. We can observe this in Figures 3.1 and 3.2. For the purpose of testing the effectiveness of this caching mechanism we made use of two graphs of $n = 10^5$ vertices, a Preferential Attachment graph (see Section 2.2.2) and a 3-Regular graph. The weights we have used were $w(u, v) = d(u)+d(v)$ for the WRW. We ran the experiments 100 times for a number of steps $s = 11n$ and recorded the time taken for each 1000 steps. We then averaged the times over all 100 experiments.

The Preferential Attachment graph has a degree distribution which follows a power-law. According to the stationary distribution of the vertices seen in Equation (3.28) higher weight vertices are expected to be visited more often, this explains the fact that (as seen in Figure 3.1) the dynamic caching mechanism is faster than pre-processing the entire graph at first, but converges to the pre-process time as it discovers more vertices. We can also observe the overhead which is introduced by a WRW when we compare with

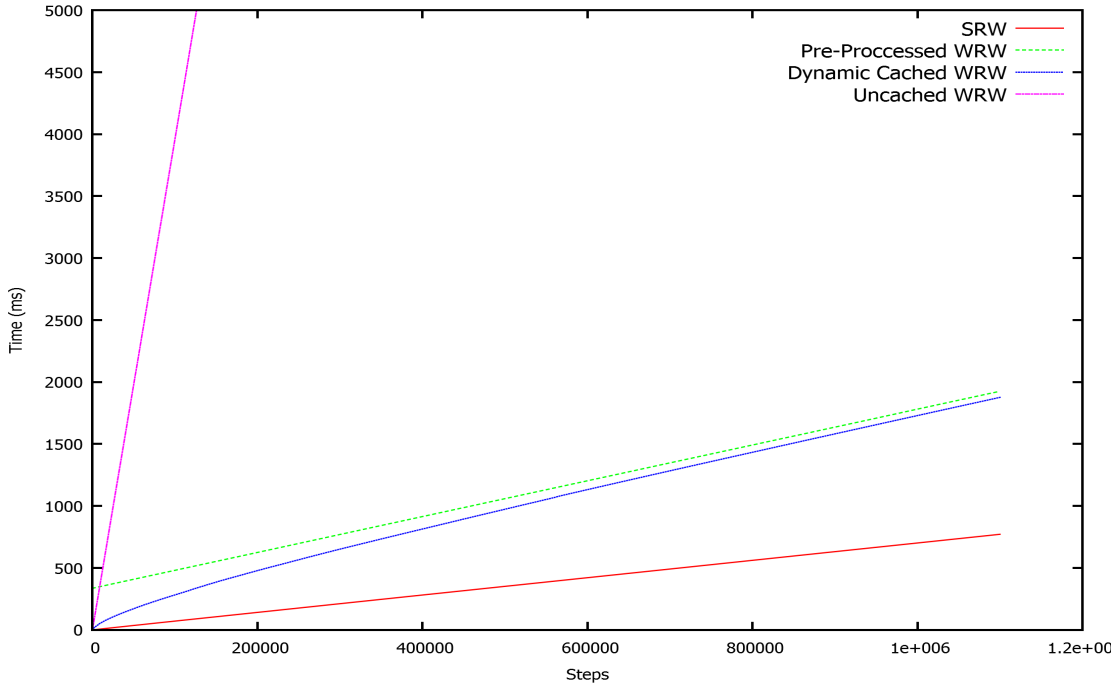


Figure 3.1: Times taken for SRW and WRW and with and without caching on Preferential Attachment Graph

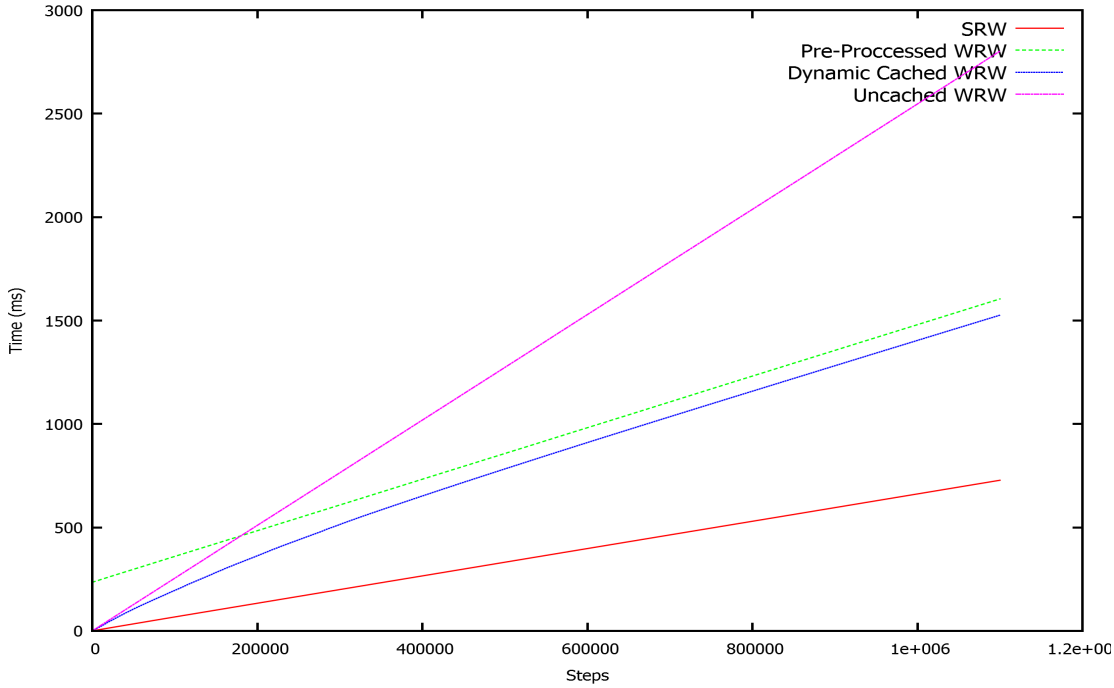


Figure 3.2: Times taken for SRW and WRW and with and without caching on d-Regular Graph

the SRW time which grows in a much slower rate. Additionally we note the times of not caching the weights at all is not fully included in the plot due to the fact that the plot line grows in a much faster rate than the other 2 lines. We have found that for a number of steps $s = 11n$ the time for an unbuffered WRW was $t = 43648$ milliseconds which is around 30 times slower than the time achieved by pre-processing and caching. This may be due to the fact that the high degree vertices were visited more often and were also the most computationally expensive vertices to get weights for.

We have also used a 3-Regular graph, that is a graph with all the vertices having degree 3. This graph is different to the Preferential Attachment graph in the sense that there are no high-weight vertices but rather all vertices have the same weight. While we could have taken advantage of this fact to modify the WRW algorithm to behave like a SRW, we did not do so for the purpose of comparison. Since the degree of each vertex is low, calculating the weights of the vertices is quick, however we can still see in this case that the dynamic caching mechanism outperforms the pre-processing mechanism up to a point, and both these caching mechanisms are faster than not using any caching. These results can be seen in Figure 3.2.

3.7 Conclusions

In this chapter we have defined Markov Chains, and have mentioned their possible uses in sampling. Additionally we have seen some properties of these chains and a special case of Markov Chains which is time-homogeneous reversible ergodic Markov Chains on graphs, which are called RWs. We have described various types of RWs, such as the SRW and WRW.

In more practical aspects of sampling with RWs we have seen the algorithmic complexity of simulating these walks. We have also presented some optimization techniques we have employed to provide speed-ups to WRW sampling but at the cost of requiring additional memory. To the best of our knowledge, such caching mechanisms are usually left implicit when working with RWs. We have shown a general and widely applicable

way to implement such caching.

Chapter 4

Estimating Network Properties: Overview

Due to the explosive growth of large online networks such as OLSNs and the WWW, the area of graph sampling methods is receiving a great deal of attention, and many open questions exist. Finding effective and efficient algorithms to sample online graphs is an area of great interest, a useful tool for the community, and has many practical applications in various fields, such as social sciences, marketing and computer science. In the area of social sciences, sampling techniques have been used in order to estimate the size of the population or estimate the proportion of the population which have a certain characteristic e.g. it is impossible to exhaustively measure the height of every person in the world, but we know the distribution of heights of people from sampling and estimation. Similarly in advertising the population is sampled to estimate the proportion of the population who uses and is happy with a certain product or service.

The topic we consider, is how to efficiently estimate global graph properties such as number of edges, triangles and vertices etc. We first present some existing methods to estimate graph properties as well as their strengths and weaknesses. In some cases, we present an analysis of such methods to determine the expected value of the property, and the accuracy of the estimates. We also present some practical applications on synthetic and real graphs.

Generally speaking, graph sampling is an underdeveloped topic. Put simply, the big question is: 'How can one sample only a part of a graph and infer properties which hold for the entire graph'. This question has many interpretations and shades of meaning. In our case, for example, we are interested in getting specific global properties of a graph which are distributed among the vertex set or a subset of the vertex set In particular we want the degree distribution that is observed in the entire network to be, at scale, observed in our sample of the network. Other typical examples might be global clustering coefficient.

Work on efficient sampling of network characteristics arises in many areas. In the context of search engine design, studies in optimally sampling the URL crawl frontier to rapidly sample (e.g.) high page-rank vertices based on knowledge of vertex degree in the current sample can be found in e.g. [3].

Within the random graph community, *traceroute sampling* was used to estimate cumulate degree distributions; and methods of removing the high degree bias from this process were studied in e.g. [1], [49]. Another approach, analysed in [15], is the *jump and crawl* method to find (e.g.) all very high degree vertices. The method uses a mixture of uniform sampling followed by inspection of the neighbouring vertices, in a time sub-linear in the network size.

In the context of online social networks, early explorations often focused on how to discover the entire network more efficiently. Until recently this was feasible for many real world networks, before they exploded to their current size. It is no longer feasible to get a consistent snapshot of the Facebook network for example. According to online sources[1], there were around 10^9 active Facebook users. While there is no way to confirm the accuracy of this statement directly, it seems to be a widely accepted estimate.

In our specific area of interest, large online networks, and in particular scale-free networks, sampling is very useful. Such networks are very large with the number of vertices ranging anything between a few thousands up to a billion vertices. Due to the sheer

[1] `http://www.statisticbrain.com/facebook-statistics/`, retrieved 8 May 2014

volume of these networks, and the fact that we do not have the time or resources to exhaustively investigate them in their entirety we look for quick and efficient tools for investigating network properties, without needing the entire network. Sampling comes as a natural step towards this goal.

Some work has been done by F. Wu *et al.* [46] and their work has provided us with the framework for the basic definitions and the methods of sampling. For each method several properties the authors identified as important were measured including:

Efficiency. How fast new vertices are discovered

Sensitivity. How much the results vary depending on the target network and the percentage of protected users within that network.

Bias. How different statistical properties are distorted in the samples depending on the sampling method.

The above are the general concern of graph sampling as we require our methods to be unbiased, successful and independent of the entry point.

According to a related study conducted by A. Mislove *et al.* [107] the major issues that arise in sampling, have to do with getting an unbiased sample quickly and effectively. Several methods are discussed such as Breadth-first search (BFS) and the *snowball method*. The snowball method is effectively like a BFS samples only k out-edge targets for each discovered vertex rather than all of them and rejects the rest. It was found experimentally that sampling the graph with these methods introduces a bias. According to their work additional challenges arise when the underlying network itself imposes query limitations, like not allowing retrieval of backwards links (as it is the case in Formspring) or imposing a strict limitation on the rate that data can be acquired (as in the case of Twitter or Facebook).

It is a fact that data rate limitations can be overcome by scraping the web interface of OLSNs rather than using the API. The method called *web scraping* successively

reads web pages from the OLSN web interface and strips them down to contain only the relevant information. However as it is pointed out in [57] this introduces a large overhead due to the fact that useless information such as web-headers and other hypertext elements will have to be downloaded. We must be very careful and the size of the overheads must be evaluated to determine whether or not we will achieve a higher throughput via scraping or by using the API with data rate limitations.

In [84] Leskovec suggests that the goals in sampling a network could include:

Back in time goal. Here one is interested in obtaining a sample of a network which has the same properties as that network in a previous epoch.

Scale-down goal. Here one is interested in getting a sample of the network which has the same properties as the network in the current epoch.

This is of course only some of the goals one could consider when sampling a network.

In addition to the above problem there is also the more general problem of when to stop the sampling. *When do we know that we have sampled enough of the network?* This question is very critical as it has many side-effects. What if our method is appropriate but we are stopping it too soon? What if we sampled too much and distorted our perfect image? The answer to this question is still largely an open problem. However in [57] some convergence tactics are discussed and analysed. These convergence tactics were:

Geweke Diagnostic. The Geweke diagnostic, proposed by J. Geweke [55], considers the convergence of a single Markov Chain. In [57] X is defined as sequence of samples of a metric of interest. The sequence is split into two subsequences, X_a which is typically the leading 10% of the sample sequence, and X_b, which is typically the trailing 50% of the sequence. The *z-statistic* is:

$$z = \frac{\overline{X_a} - \overline{X_b}}{\sqrt{\mathbf{Var}X_a + \mathbf{Var}X_b}} \tag{4.1}$$

where $\overline{X_a}$ and $\overline{X_b}$ are the mean values of the subsequences X_a and X_b respectively. Since X_a and X_b are samples from the same distribution then z is normally distributed with mean 0 and variance 1. In [57] it is mentioned that we can declare convergence if $z \in [-1, 1]$.

Gelman-Rubin Diagnostic. The Gelman-Rubin diagnostic, proposed by A. Gelman and D. B. Rubin [53] suggests the use of $m > 1$ parallel RWs from different starting points, to ensure that the sample means are not correlated to a specific starting position in the sample space. As mentioned in [57], this diagnostic compares the distribution of each RW with the overall distribution of all the RWs together. In [53] the convergence test is as follows:

$$R = \frac{\hat{V}}{\sigma^2} \cdot \frac{df}{df - 2} \tag{4.2}$$

where $\hat{V}$ is the estimate of the variance of the underlying t-student distribution and df is the degrees of freedom of the t-student distribution estimated by the method of moments. The value of R converges to 1 as the number of samples approaches infinity, and values close to 1 (typically below 1.02) indicate convergence.

These methods offer a great tool for the crawler to be able to determine when it is best to halt the crawling.

4.1 Estimating Population Size Using Uniform Sampling

Sampling is a broad term used to specify the act of obtaining *representative* data from a group. In the social context sampling and statistics were developed as a tool to infer trends and general behaviour of social groups without exhaustively investigating the entire group. In the context of large online networks sampling is an area of great interest, mainly due to the very large size of these networks. Obtaining network properties has a great interest on its own. However, exhaustively searching such massive networks comes at great time and space costs which are usually undesirable and sometimes impossible to

meet. Sampling is a natural tool that meets the challenge of obtaining network properties at the minimum cost possible, even if it means sacrificing some accuracy.

In a other scientific contexts, such as e.g. sociological attempting to exhaustively obtain properties of a population leads to inevitable limits to the size of population size that can be handled. An example of this can be seen in the work of W. W. Zachary [132] where a karate club was investigated exhaustively in order to observe the social fission and re-formation after conflict. The size of this karate club was no more than 100 members and larger sizes are often challenging or impossible to deal with. Due to these challenges researchers have turned to alternative ways of obtaining useful statistics of populations.

Estimating the size of a population has been very important in various contexts including sociological, anthropological and zoological. Typically in such contexts, the populations investigated are very large. In addition, exhaustive search is generally not feasible. Even if it is feasible, it is an expensive and time consuming process. These practical limitations had led to the need of estimating properties within reasonable accuracy, rather than exhaustively searching. Many methods were developed for this purpose. D. W. Hayne [63] presented some methods that were used in zoological contexts in order to estimate the population size.

We will use the following notation:

Definition 4.1 (Sampling definitions)**.** *We define the* population size n *as the number of elements in a set* $\mathbb{S}$ *such that* $|\mathbb{S}| = n$*. A sample* $S_i \subseteq \mathbb{S}$ *that is obtained from* $\mathbb{S}$ *will consist of elements of that set, e.g.* $S_i = \{p_1^{(i)}, p_2^{(i)}, \ldots, p_k^{(i)}\}$ *where* $p_j \in \mathbb{S}$ *and the size of the* i*-th sample is referred to as* $|S_i| = k_i$*. We refer to an element* $p_j \in S_i$ *as the* j*-th sample point in the* i*-th sample. We will use the notion of* collision *to denote a pair of sample points* $(p_j^{(i)}, p_l^{(i)})$ *where* $j \neq l$ *and* $p_j^{(i)} = p_l^{(i)}$ *i.e. when during a sampling we sample the same point twice.*

4.1.1 Moran-Zippin method

This method of population size estimation again comes from the field of zoology and specifically on the work of C. Zippin [136] and P. A. P. Moran [110]. The method is as follows: Let n be the population size (unknown). Since this method is from a zoological

context, the sample size is generally not something that can be controlled. The sampling takes place in a fixed time frame, and the population that can be sampled within that time frame varies. Assuming two consecutive samples without replacement from the same population S_1 and S_2 with $|S_1| = k_1$ and $|S_2| = k_2$ then the size of the set which remains after the first sample should be $n - k_1$. The proportion of the set which was sampled during the second sampling is $\frac{k_2}{n-k_1}$. The assumption here is that the two samples obtained the same proportion of the population or $\frac{k_1}{n} = \frac{k_2}{n-k_1}$ and therefore the estimate would be

$$\hat{n} = \frac{k_1^2}{k_1 - k_2}$$

4.1.2 Lincoln-Petersen method

Another method used to estimate population sizes is the Lincoln-Petersen method or Lincoln index. This method was first described by F. C. Lincoln [93], however it was first used by C. G. J. Petersen [116]. Similarly to the Moran-Zippin method described in Section 4.1.1, this method also requires only two samples to be collected. Again, the samples are taken in a fixed time frame and vary in size. The method works under the assumption that both samples taken are *independent and identically distributed* (*i.i.d.*). The sampling done in this case is with replacement, i.e. after the first sample is taken, all sampled points are returned to the population. Therefore there may exist points $p_i^{(1)} = p_j^{(2)}$ i.e. points which are sampled during both samplings. The assumption here is that the proportion of previously sampled members of the population, would be representative to the overall population. If for example p is the proportion of the population sampled during the second visit, i.e. $p = \frac{k_2}{n}$ then this should contain a p proportion of previously sampled members of the population i.e. $\frac{|S_1 \cap S_2|}{k_1} = p$. By solving for n we derive the the Lincoln-Petersen estimate, which is:

$$\begin{aligned} \frac{k_2}{n} &= \frac{|S_1 \cap S_2|}{k_1} \\ \hat{n} &= \frac{k_1 k_2}{|S_1 \cap S_2|} \end{aligned} \tag{4.3}$$

While this method is asymptotically unbiased for large sample sizes, it is biased for small sample sizes [19]. An alternative approach, which is less biased, would be the Chapman estimator [19]. This estimator based on the same sampling procedure, but the estimate is given by:

$$\hat{n} = \frac{(k_1 + 1)(k_2 + 1)}{|S_1 \cap S_2| + 1} - 1$$

4.1.3 Regression method

The *regression method* is a method for estimating population size which has been widely used in zoological contexts such as e.g. in the work of G.G. Marten [99]. A graphical example of this method can be seen in Figure 4.1. While this method is widely applied in zoological contexts it is generally applicable. Using the notation as defined in Definition 4.1 we proceed to describe the regression method.

The initial assumptions of the method are that sampling is done with replacement. The procedure for the regression method is as follows:

- Assume we sample with replacement obtain k points from set $\mathbb{S}$. We then remove any points we have previously sampled to obtain sample $k_1 \approx k$.
- We repeat this process to get r disjoint samples of sizes $k_1, k_2, \ldots, k_r$ respectively.
- We plot the number of samples k_i against the total number of samples obtained during all previous times i.e. $c_i = \sum_{j=1}^{i-1} k_i$.
- In the simplest form of linear regression we would visually inspect these elements and draw the line which would pass through a minimal distance from all of them.
- We would then extrapolate to find the point of intersection of that line with the x-axis. Presumably the point where $y = 0$ would be the point where $x = n$

The intuition of the method is that if we continue to sample we eventually would have sampled the entire population. The underlying assumptions here are that if the sample

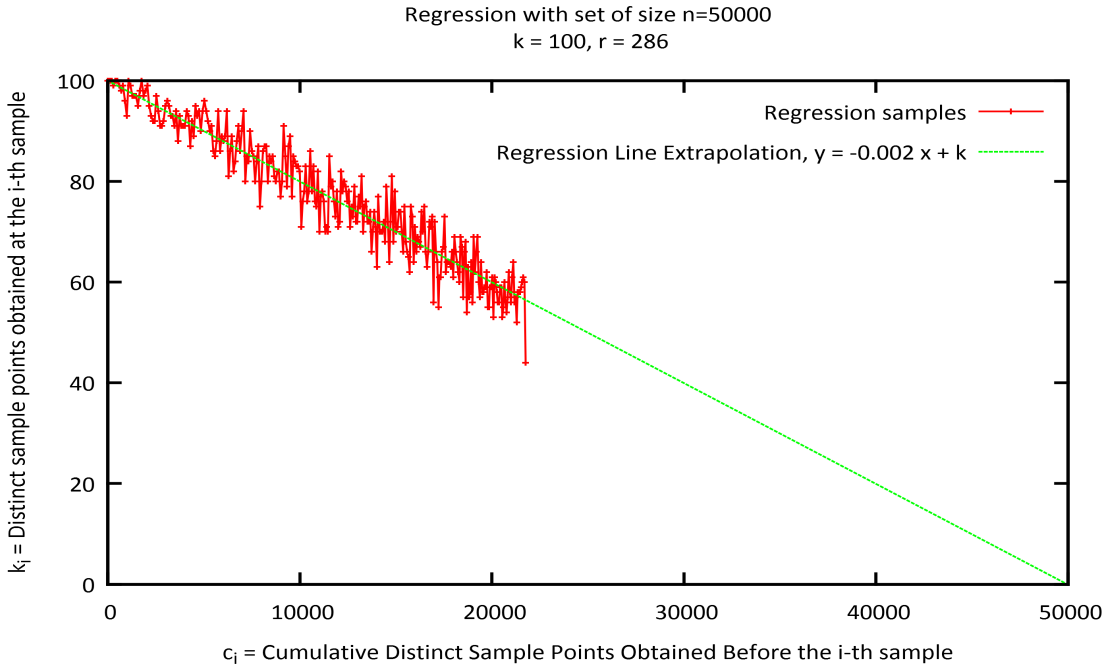


Figure 4.1: Population estimation using the regression method. $n = 50000$

probability of each point is uniform then the rate in which new points are obtained would decrease linearly w.r.t. the number of distinct points that have been sampled. This will continue to occur until no more new samples can be obtained. Since this decline is linear then the point of intersection of the line drawn with the x axis should be the size of the population since at the point the y-axis is 0 then the x-axis should be n. This can be seen in Figure 4.1 where the extrapolation of the line which best matches the data passes approximately through n.

A simple intuitive explanation of why this works would be the following: For the sake of simplicity we will denote with c_i the cumulate sum of previous samples i.e. $c_i = \sum_{j=1}^{i-1} k_i$. The following two properties hold: $k_1 \approx k$ and $c_1 = 0$. For the i-th sample we obtain k points, however for each point there is a $\frac{c_i}{n}$ probability we have sampled it before and therefore the expected k_i would be: $k_i \approx k - \frac{kc_i}{n}$. In Figure 4.1 the y-axis would correspond to the number of successful samples which would be k_i while the x-axis

would be the previous samples obtained c_i. Therefore two points (c_i, k_i) and (c_j, k_j) would define a line with slope:

$$\begin{aligned} m &= \frac{k_j - k_i}{c_j - c_i} \\ &= \frac{(k - \frac{kc_j}{n}) - (k - \frac{kc_i}{n})}{c_j - c_i} \\ &= \frac{\frac{kc_i - kc_j}{n}}{c_j - c_i} \\ &= \frac{k}{n}\frac{c_i - c_j}{c_j - c_i} \\ &= -\frac{k}{n} \end{aligned} \tag{4.4}$$

Assume point $(0, k)$ our first point in the regression line, then the line which would pass through this point and has slope m would be given by: $y = mx + k = -\frac{k}{n}x + k$ and therefore when $y = 0$ the value of x would be n. This would only work assuming that during the first sample $k_1 \approx k$ which means that there are not many collisions within the sample. This would mean the size needs to be small e.g. $k < \sqrt{2n}$. The reason for this particular value is discussed in Section 4.1.5. We note that this explanation is not a formal proof since there are many other aspects to take into account, such as variance and the effect that $|k_1 - k|$ has on the estimate. In the case of Figure 4.1, the slope was was derived by fitting the data with the line $y = mx + k$ where m is the slope of the line. The fitting of the data was done using the built-in Gnuplot function `fit`[2] which uses an implementation of the non-linear least-squares (NLLS) Marquardt-Levenberg algorithm [90]. The value of the slope which was calculated using this method was $\hat{m} = -0.00200157$ which is very close to the theoretically expected $m = -\frac{k}{n} = -0.002$.

4.1.4 Coupon collection

Coupon collection is a well known technique. More on this problem can be found in Chapter 7 and 15 of *Problems and Snapshots from the World of Probability* [10]. The problem definition states: Assume there is a set of items $\mathbb{S}$ of size n. We wish to sample

[2] http://gnuplot.sourceforge.net/docs_4.2/node82.html, Retrieved 27/06/2014.

this set *uar* until we have observed all the elements contained in it i.e. until we have collected all elements. How many times would we need to sample items from $\mathbb{S}$ *uar* in order to collect all of them?

Assuming after sampling k elements *uar* we have a sample set S which contains $|S| = i-1$ distinct samples elements then the probability that our next sample s_{k+1} will obtain an uncollected element is

$$P(s_{k+1} \notin S \mid |S| = (i-1)) = P_{i-1}(s_{k+1}) = 1 - \frac{i-1}{n}$$

If we consider $P_{i-1}(s_{k+1})$ to be to probability of "success" then the expected time Y_i until success follows a geometric distribution with probability $P_{i-1}(s_{k+1})$ which, has an expected value $\mathbf{E}Y_i = \frac{1}{P_{i-1}(s_{k+1})} = \frac{n}{n-i+1}$. The expected time to collect all items would be

$$\sum_{i=1}^{N} \mathbf{E}Y_i = n(\frac{1}{N} + \frac{1}{N-1} + \ldots + 1) = nH_n$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th *harmonic number*.

The coupon collection method is not typically viewed as a population/set size estimator. On the other hand the expected number of samples to collect all items coincides with the cover time of RWs on the complete graph. Generalizing the above equations we can easily obtain the cover times of RWs for graphs with a near-uniform stationary distribution, assuming the RWs have a small mixing time τ_2. The reason we require a small mixing time is due to the fact that while the stationary distribution may be near-uniform, the distribution of samples taken before the RW is stationary are not necessarily uniform, and in most cases are correlated to the starting point.

If our sampling is not uniform but taken from some distribution $f(x)$ then the general formulae are:

$$P_{i-1}(s_{k+1}) = 1 - \sum_{s \in S} f(s) \tag{4.5}$$

Giving a generalized expected waiting time Y_i

$$\mathbf{E}Y_i = \frac{1}{1 - \sum_{i=1}^{i-1} f(s_i)}$$

The overall waiting time until all elements are collected would be

$$\sum_{i=1}^{N} \mathbf{E}Y_i = \sum_{i=1}^{N} \frac{1}{1 - \sum_{j=1}^{i} f(s_i)}$$

4.1.5 Sample-and-collide method

This method is a generalization of one form of the *birthday paradox* which states that if we assume that the birthdays of people are uniformly distributed and if we sample uniformly from a population of people then the expected number of samples required before we sample an individual with the same birthday is $\sqrt{2n}$ with $n = 365$ in this case. The paradox is that the actual theoretical and experimental result does not satisfy the general intuition in which we would expect a collision at a sample size of $\frac{n}{2}$. While this has no theoretical support, the name "paradox" has remained for this problem, for historical reasons. In order to generalize this problem, assume we obtain k *uar* samples from a set $\mathbb{S}$ of size n. There are two properties which we can consider:

1. How many samples k do we need to get until we get a repeating sample (collision)?
2. How many collisions are we expected to have in a sample of size k?

This problem is an interesting one and has been generalized by McKinney [103] where it was shown that, the expected number of repetitions in k samples is $k(k-1)/2n$ with variance $(1 - 1/n)\ k(k-1)/2n$. The use of this method to estimate network size is described by Bawa *et al.* [8].

The method of sample and collide requires *uar* samples with replacement from the population. Assume that we successively sample a population $\mathbb{S}$ until we find a member of the population twice. Since we are sampling *uar* then the probability of sampling a particular member is $\frac{1}{n}$ where $n = |\mathbb{S}|$.

Assuming a sample $S_1 = \{x_1, x_2, \cdots, x_k\}$ of size k taken uniformly at random from $\mathbb{S}$, if k is the sample size when the first repetition occurs, then an estimate of the network size

is $\widehat{n} = S_1^2/2$. Explanation follows: The number of collisions in the sample are expected to be:

$$\mathbf{E}C = \binom{k}{2} \sum_{i=1}^{n} p^2 = \binom{k}{2} \frac{1}{n} \tag{4.6}$$

which means that we expect 1 collision when we have a sample of size:

$$k = \sqrt{2n}$$

In this case is to get the average sample size $\overline{k}$ required for a single collision and get an estimate of $\hat{n} = \frac{\overline{k}^2}{2}$.

4.2 Obtaining uniformly at random samples using random walks

The methods described in this chapter rely heavily on the acquisition of uniform samples, however, obtaining such samples is in practice very difficult. Large online networks generally do not support this sort of random access, in many cases due to privacy concerns. However we will describe some ways we could "work around" these limitations.

Sampling the elements of a set uniformly at random with replacement can be used to estimate the set size in sub-linear time by the method of *sample and collide*, described in Section 4.1.5. However, the method of sample and collide requires *uar* samples from the population.

To obtain a uniform sample from a network using a random walk, we can do the following: Run the walk for $T \geq \tau_2$ steps before sampling, where τ_2 is a suitable mixing time. In this case the walk is in near stationarity, and $P_u(X_t = x) \sim \pi_x$, where X_t is the position of the walk at step t, and π_x is the stationary distribution of the walk. For a simple random walk $\pi_x = d(x)/2m$, where $d(x)$ is degree of x, and m is the number of edges. Thus, unless the graph is regular $\pi_x \neq 1/n$ and so the sample is not uniform. To use a sample from the stationary distribution and we need to unbias the walk. There are several ways to do this.

4.2.1 Using a continuous time random walk as a surrogate for uniform sampling.

We review the approach of Massoulie *et al.* [100], and Ganesh *et al.* [52], who use a continuous time random walk with a fixed stopping time.

The method suggested by L. Massoulié *et al.* [100], is based on a continuous time random walk. This walk makes use of continuous time to unbias a simple random walk.

The sampling algorithm in [100] proceeds as follows:

1. A timer is set at some predefined value $T > 0$, by the initiator, node u, in a sampling message.

2. Any node v, either after receiving the sampling message, or (if it is the initiator) after having initialised the timer, does the following. It picks a random number R, uniformly distributed on $[0, 1]$. It sets $T' = T + \frac{\log(R)}{d_i}$. If $T' \leq 0$, then this node v is the sampled node; it returns its ID to the initiator, and the procedure stops. Otherwise, it forwards the message with the updated timer T' to one of its $d(v)$ neighbours, chosen uniformly at random.

This procedure returns a random node sample, the distribution of which is exactly that of the state of the standard CTRW at time T, started from node u. The reason is that, since U is selected *uar* from $(0 \ldots 1]$ and therefore has an exponential distribution with $\mathbf{E}\log(R) = -1$. Using the procedure above, the claim is that each sample gathered is an approximate uniform sample. This can be implemented as a simple random walk, either by holding the simple walk at u for a time exactly $\frac{1}{d(u)}$, or by holding the walk for an exponential waiting time with mean $\frac{1}{d(u)}$, such as e.g. $-\frac{\log(R)}{d(u)}$.

Assume we run the CTRW described above, where at each step, we hold the walk at vertex u for an expected time of $\frac{1}{d(u)}$. Assume the total time the CTRW has been held is T. F or each vertex u, the expected proportion of T for which the walk was held at u is $\frac{T\pi_u}{d(u)} = \frac{T}{2m}$. Since the walk is held at each vertex for time $\frac{T}{2m}$ which is uniform among all

vertices, then for the probability of the walk being at vertex u at any given time $T' \leq T$ is $P(X_t = u \mid T') = \frac{T}{2m}$.

4.2.2 Using a Re-Weighted Random Walk

An alternative way to collect uniform samples is by using a Re-Weighted Random Walk (RWRW).This is used to unbias random walks and has been widely used e.g. in [57]. The general idea is based on *rejection sampling* [126]. Rejection sampling is the method that states that if we want to sample from a specific distribution $f(x)$ but are unable to, we may instead choose to find another distribution $g(x)$ which we are able to sample from. If there exists a constant ℓ such that $\ell g(x) \geq f(x) \forall x$ then accepting samples drawn from $\ell g(x)$ with probability $\frac{f(x)}{\ell g(x)}$ will result in the samples accepted to follow distribution $f(x)$.

In this case if we run a SRW for time $T \geq \tau_2$ then the probability of sampling v would be $P(X_T = v) = \pi_v = \frac{d(v)}{2m} = g(v)$. On the other hand we with to sample v *uar*, that is (ideally) with probability $P(v) = \frac{1}{n} = f(v)$. Assuming the graph is sparse then $m = cn$, therefore $g(v) = \frac{d(v)}{2cn} = \frac{d(v)}{2c} f(v)$ therefore setting $\ell = \frac{d(v)}{2c}$ we have (from e.g. [126]):

$$\begin{aligned} P(accept \mid v) =& \frac{f(v)}{\ell g(v)} \\ P(v) =& g(v) \\ P(accept) =& \frac{1}{\ell} \\ P(v \mid accept) =& \frac{P(accept \mid v) \times P(v)}{P(accept)} = f(u) \end{aligned}$$

We remind that $\ell = \frac{d(v)}{2c}$, however c is not generally known. On the other hand we only need to satisfy the requirement that $\ell g(x) \geq f(x)$. Since $\frac{d(v)}{2} \geq \ell$ as $c \geq 1$ we can set $\ell_1 = \frac{d(v)}{2}$. We can use $P(accept) = \frac{1}{\ell_1} = \frac{2}{d(u)}$ to obtain uniform samples. In this case, this may change the rate of rejection, and therefore the number of steps we need to take in the underlying random walk. We note that we can implement the same strategy using a WRW by replacing $d(v)$ with $w(v)$ and using the WRW stationary distribution. However it is more likely we know the mixing time τ_2 for a SRW rather than a WRW.

4.2.3 Using a Metropolis Hastings Random Walk

An alternative method is to use a MHRW random walk with target stationary distribution $\pi_x = 1/n$. One problem with this approach, and indeed all random walk based sampling, is that the mixing time τ_2 is often unknown. This becomes even more apparent in the case of the MHRW since, unlike the RWRW, the steps in which the walk does not make any transition, greatly increase the mixing time. If we view the mixing time as the time required for the walk to "forget" where it started from then the longer the walk position is correlated to the starting vertex, the more steps would be required for the walk to reach stationarity.

We remind that the acceptance probability of a MHRW designed to collect uniform samples is given in Equation (3.35), namely, $a(u,v) = \min\left(1, \frac{d(u)}{d(v)}\right)$ if $u \neq v$.

An alternative method would be to relax the acceptance probability and set the transition probability matrix as follows:

$$p(u,v) = \begin{cases} \frac{1}{M}, & \text{if } v \in N(u), \\ 1 - \frac{d(u)}{M}, & v = u \\ 0, & \text{otherwise} \end{cases}$$

where $n \gg M \geq \Delta(G)$ the maximum degree of G. This is equivalent to making the graph M-regular by adding loops on vertices with degree lower than M.

4.3 Uniform Sampling from the degree distribution

In the initial steps of our research we had experimented on crawling the Twitter OLSN. The initial approach of the crawling method was a simple random walk which was performed on a single lab computer. The inherent limitation was that the Twitter API only allowed 150 requests per hour. The way that the API was utilized in this initial phase was, starting from a specific vertex, to retrieve all the in and out-links and then randomly select one of those to continue with the crawl while rejecting all the previously visited ones. This method is known as *exploration without replacement* or *graph traversal*.

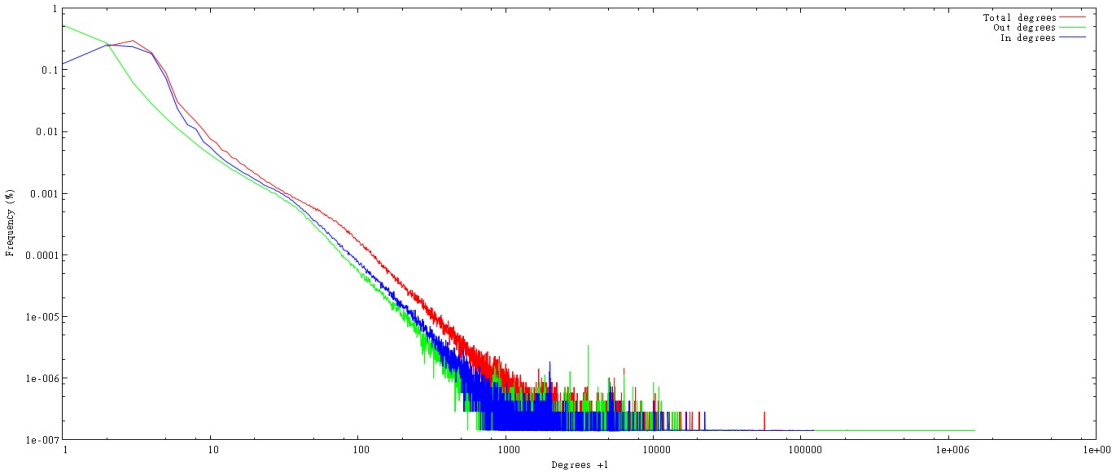
The notion of in-degree and out-degree of the Twitter graph is ambiguous and depends on the point-of-view of the network. We define this as the direction of the flow of information, in our case "tweets" where a user's out-degree is the number of followers of that user, who are recipients of tweets from the user, while the user's in-degree is the number of users that user is following i.e. the number of people that user is receiving tweets from.

The above process had many challenges, namely the limitation of the API to only receive 5000 in or out links per request and the requirement that the request for in-links and out-links to be done as separate API calls. We present the example of a well known Twitter site of the comedian Stephen Fry who approximately had 50,000 friends and 2,500,000 followers at the time. In order to fully acquire the friends and followers of this specific user, due to the API limitation we would have required 520 separate requests which would take around 4 hours to complete.

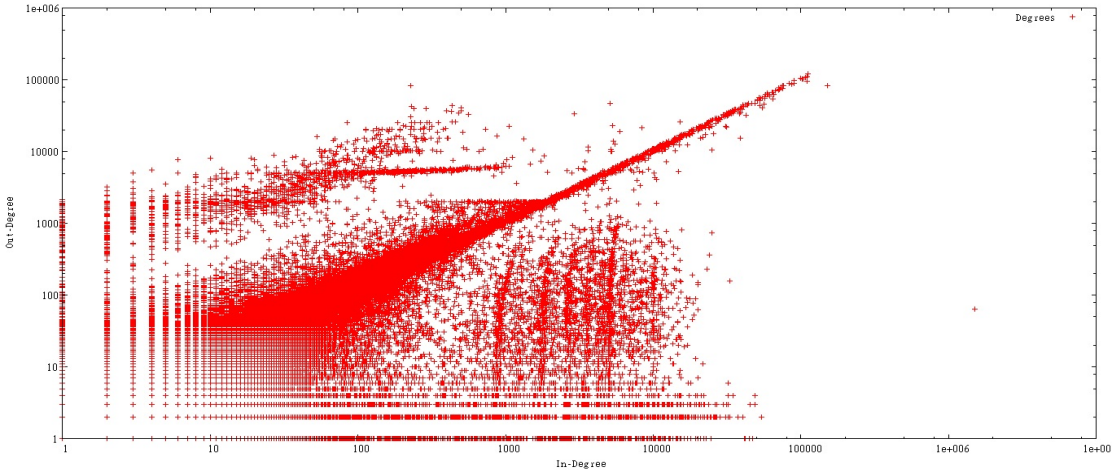
Due to the inherent bias in the random walk process we were very likely to encounter more high degree vertices as a consequence of Equation (3.28). Such vertices would take hours to fully crawl. This resulted that in a period of two weeks we only fully crawled around 5000 users. However this gave us a strong indication of the small world structure of Twitter as well as its expansive nature. We note that the 5000 users not only were well connected between them, but we also determined that the combined in and out edges of those 5000 users were over 20 million. The results of this crawling are further analysed in Section 4.3.1.

4.3.1 Twitter: Initial crawling

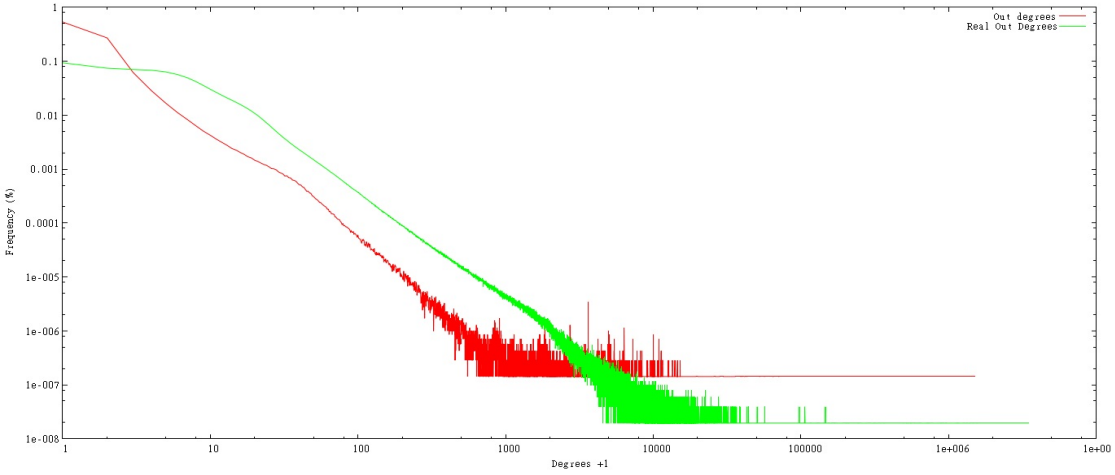
As it has been mentioned we crawled a much smaller part of the Twitter network using a sampling method without replacement. Generally methods without replacement or graph traversals are methods which do not visit the same vertex multiple times. The deduced sub-graph of Twitter we have acquired using this crawling had the properties presented in Figure 4.2. We have used a previously collected snapshot of Twitter by M.



(a) Crawled Data: Total(red) in(blue) and out(green) degree distributions



(b) Crawled Data: In-degree to Out-Degree Observations



(c) Out-degrees. Real data(green) and Crawled data(red)

Figure 4.2: Real Twitter data vs crawled data

Cha *et al.* [18] who collected the entire Twitter network of November 2009. Since this dataset was unbiased, and based on the assumption that Twitter is scale-free, we found it reasonable to assume that the data from [18] were representative of the actual Twitter data. A more detailed investigation on this can be found in Section 4.3.2.

In Figures 4.2 we can observe the relative bias of the random walk method especially when observing 4.2c. As we can see the plot of the crawled data tends to approach the real data on higher degrees, however on lower degrees the frequency is much lower than the real frequency. Since the high degree vertices are generally very few, we would not expect to discover as many if we were to sample *uar*. It is this observation, when combined with the results presented in Section 4.3.2 which have provided us with a motivation to further investigate the efficiency of sampling *uar* in scale-free graphs.

Additionally its worth noting that the relative bias of our crawling was small in the lower degrees. This may be in part a consequence of the fact that the above data consist of the merging of several random walks. Each different random walk started from the same location and crawled the graph for approximately the same amount of time. The results of each crawler were merged to create the graph image presented in Figure 4.2c.

The problems with the initial crawling, were not only the API limitations but also the hardware limitations we were imposed with. Generally there were two options available for storing the intermediate graph: either in the main memory or on the hard drive. The first case had the limitation of not being able to store a large amount of data (due to the fact that the memory of the computer used was limited) and in the second case proved to be infeasible to use, due to the high access time of the hard disk.

4.3.2 Second crawling, unbiased sampling

Our second crawling was mainly focused on discovering the actual degree distribution of the Twitter network. This research happened at a time after we had analysed the real Twitter data from [18] and had a good idea how the network looked like in 2009. This however was not enough, because we wanted to know if the structure of the network had

remained unchanged, at least in the aspect of the degree distribution. Due to this we started a second crawl of the Twitter network, this time using uniform sampling.

In order to do our sampling we took advantage of the fact that each Twitter user, when created, is assigned a unique user ID. To the best of our knowledge and backed by observation of a subset of users, the user Ids are assigned incrementally according to the order each user was created. This suggested that there was a very strong possibility that if we generated a random number ranging from the smallest to largest observed user ID at the time we would have a good possibility to have obtained a valid user ID. The method we used was to generate random numbers between 15 and $300,000,000$ which were the smallest and largest Twitter ids at the time. We then performed a user info request from the Twitter API which is a single request that returns a great deal of data regarding Twitter users such as in-degree, out-degree, number of tweets sent, age of the account etc. This request, while not providing the actual in-edges and out-edges (i.e. the other endpoints) did provide us with a good idea regarding the degree distribution.

We have observed that the actual valid user Ids that fell within our sample range were around 75% of all the Ids tried, which seems to follow our intuitive assumption. The above method of rejecting a proportion of the generated samples is a method of rejection sampling and had been used in the past by e.g. [82].

In the process of this crawling we gathered data regarding 378303 users and the resulting degree distributions are presented in Figure 4.3.

From what is seen in Figure 4.3 we can determine that the degree distribution follows a power-law for both the in and out-degrees. In addition to this the power-law coefficient seems to be the same for both distributions and also for the total degree. A unique observation in this plot is that the number of statuses (or tweets) that users have sent also follow a power law. To elaborate what exists in the above plot. Let $f(x)$ be the number of users who have sent x tweets and n the total number of users then what is plotted on the above is $t(x) = f(x)/n$. This distribution seems to be similar to the degree distributions with a lower slope. A rough estimate of the power-law in the mid-

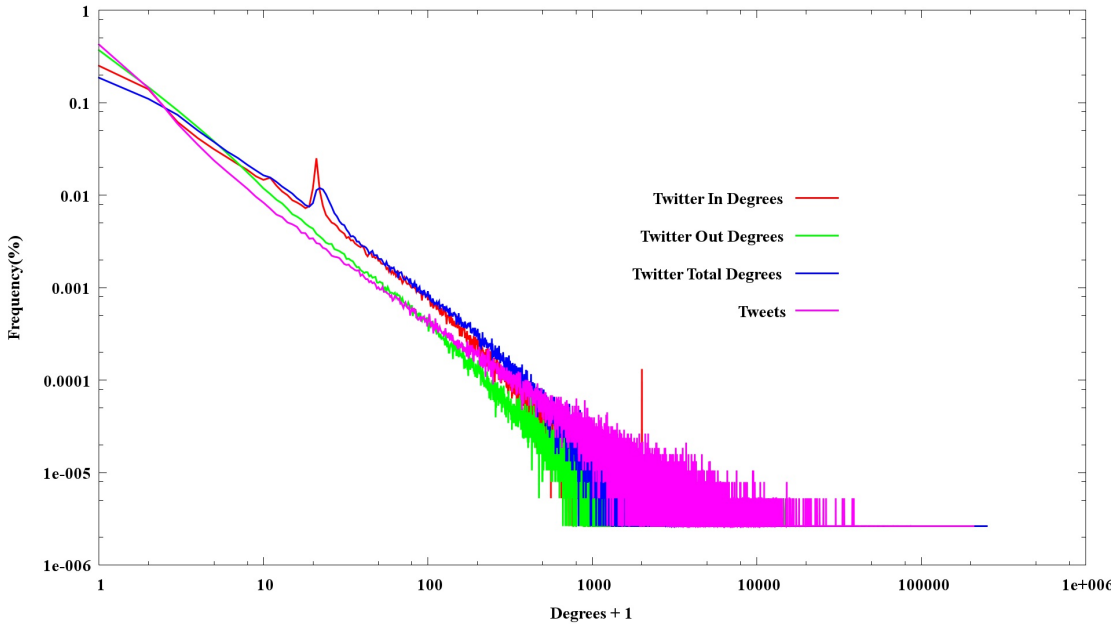


Figure 4.3: In-Degrees (red) Out-Degrees (green) Total Degrees (blue) and tweets (fuchsia) as a function of frequency on a log scale

range indicates that it has a co-efficient of $c \approx 1.6$ which is significantly lower than what the currently analysed generative models are able to reproduce.

This data also gave us the opportunity to make some reasonable comparisons to the dataset from the 2009 Twitter crawl. The result of this comparison can be seen in Figure 4.4. We note that only the results of the in-degree comparison are presented due to the fact that they include a certain anomaly which we hoped to examine and determine whether that was an artefact of the crawling or of the network itself. The comparisons of the other distributions has similar results.

From Figure 4.4 we can observe that the degree distribution is the same, however this cannot be said with certainty as the scale of the two data sets is largely different. It is worth mentioning that the large data-set from [18] does not include vertices of total degree 0, which seem to take up a significant portion of the network. Also when viewing both Figure 4.3 and Figure 4.4 we can see anomaly when the in-degree is 20. This seems to be artefact of the Twitter network and appears to have remained. From empirical

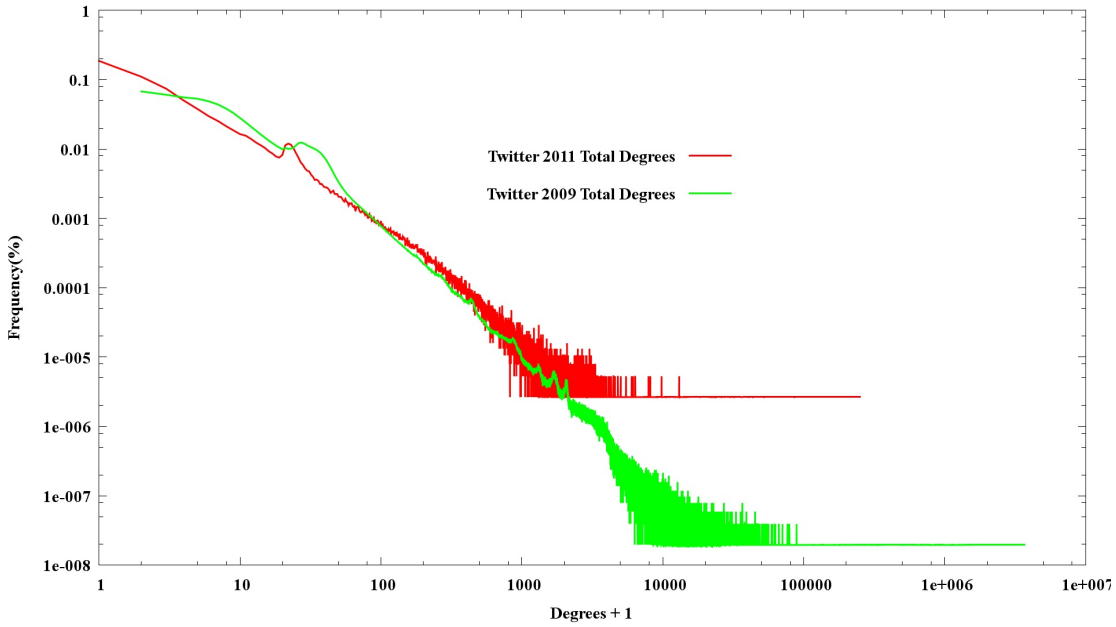


Figure 4.4: Real 2009 Twitter distribution (Green) compared to sampled Twitter distribution (Red)

observation a number of the users with in-degree 20 do not correspond to people but certain services, sometimes malicious. It seems that there might be a limit imposed either by Twitter or by other limitations which restricts these kinds of accounts to an in-degree of 20.

All the above have led us to the assumption that the actual Twitter network today, while being much larger, has a similar degree distribution compared to the network as it was in 2009, however it seems that the slope is slightly different.

4.3.3 Uniform Sampling: A study of efficiency

In this section we make a small case study of uniform sampling. We have performed some tests by sampling vertices with respect to degree *uar* from a preferential attachment network. This study was made in order to correctly and accurately measure the efficiency of the uniform sampling method depending on the size of the sample. The resulting image was mostly what was expected (which was a near accurate degree distribution sample even for small sample sizes) however we did make some important observations.

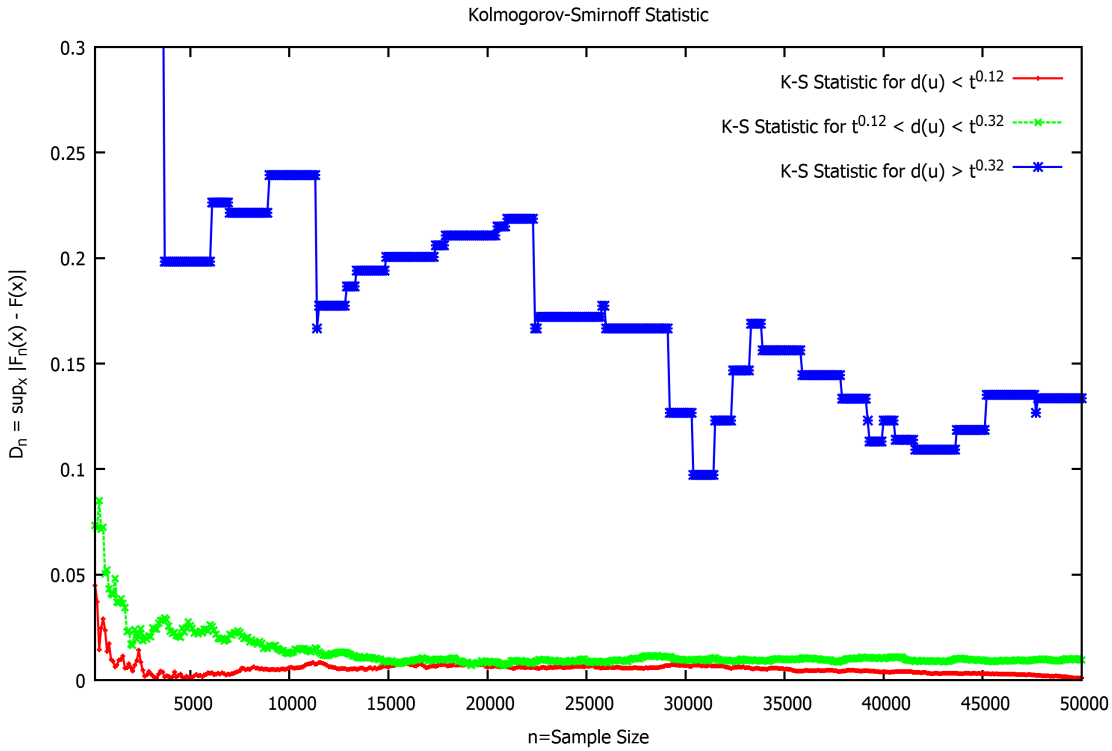


Figure 4.5: K–S test on a preferential attachment graph

We will present our findings here but before doing so we will describe the method we used to test the efficiency of sampling *uar*. The method we used was the widely used and described Kolmogorov-Smirnov Test (K–S test) [108]. This goodness of fit test is defined as follows:

Assuming that we are sampling from a distribution with c.d.f. $F(.)$. We use $F_n(\cdot)$ to denote the c.d.f. of the *empirical distribution*, i.e., our sample for n *i.i.d.* observations. Then the K–S test is defined as:

$$D_n = \sup_{-\infty<x<\infty} |F_n(x) - F(x)| \tag{4.7}$$

This measure is used with a specific *null hypothesis* and tests whether the sample confirms this hypothesis if it falls within specific confidence intervals. In our case, the hypothesis was that the sampled degree distribution c.d.f. would converge to the real c.d.f. of the degree distribution which we could measure beforehand in our manufactured

graph. Under this hypothesis $F(x) = P(d(u) \leq x)$ the probability a vertex has degree less than x while $F_n(x)$ is the actual proportion of sampled edges of $d(u) \leq x$.

The results of the above test are shown in Figure 4.5. What is shown, is three different K–S tests. Each of those tests, uses the null hypothesis that the sample is taken from a c.d.f which includes the degree distributions of specific degree ranges. This corresponds to the 3 areas of the degree distribution which can be seen in Figure 2.2. These ranges are: (a) degrees $d(u) < n^{0.12}$, (b) degrees between $n^{0.12} \leq d(u) < n^{0.32}$ and (c) degrees $d(u) > n^{0.32}$. Under the K–S test, for a sample of size n the null hypothesis is rejected at level a (i.e., is rejected with confidence a) if

$$\sqrt{n}D_n > K_a$$

where K_a is given by the probability $P(K \leq K_a) = 1 - a$. In this case K is the Kolmogorov distribution [98]. The values of K_a are given by tables of critical values (such as the one found in e.g. [70]).

The reason we decided to partition the tests in cases was the nature of the distribution; there was a need to perform this sort of separation in order to determine how well a uniform sampling process would perform in finding *authorities* in a real network situation. As we can see in Figure 4.5 the sampling converges much slower for the higher degree vertices. The sampling is very efficient even at very small samples in the low and medium range ($d(u) \leq n^{0.32}$). Indeed, sampling *uar* is not a good method to sample the c.d.f. of large degrees because there were few such vertices. This important observation is what led to the inspiration of a new method which would perform this sampling better. This method is based on a WRW (see e.g. Section 3.3) and will be described in Chapter 7.

4.4 Conclusion

In this chapter we note the importance of estimating various properties in populations. We show the work that had been done in the past to estimate sizes of populations in various contexts. In our specific area of research we typically represent these populations

as graphs, and specifically target large online networks. Now the notion of population is not very specific in our case but can rather correspond to (a) people (b) web-pages (c) autonomous systems (d) agents etc. We show that uniform sampling has been an important tool in determining various properties of these populations and show various methods we can simulate uniform sampling if uniform sampling is not directly possible on the targeted graphs.

We also present some results comparing a snapshot of the the Twitter network as well as samples taken either uniformly, or by using a SRW. We show that sampling *uar* works well in practice in the majority of the network, however is not very successful in sampling under-represented data, such as high degree vertices, which are rare in the targeted graphs.

Chapter 5

Estimating Network Properties Using Random Walks

In Chapter 4 we have described the need for sampling and estimation of certain network properties such as e.g. the size of the network. There are certain drawbacks on these methods, namely they require uniform sampling of the vertex set of the graph. In Section 4.2, we have seen some methods to obtain uniform samples from a graph indirectly using RWs. In this chapter we present four RW based methods to obtain global network properties.

The ideas developed are joint work with C. Cooper and T. Radzik and appear in [35, 38, 39]. We develop the following ideas, both theoretically and experimentally.

1. Global properties of graphs can be estimated using first return times of RWs.
2. For a given property, vertex weights based only on local structure can be designed which return the desired estimate of the global property.
3. Vertices with the highest weight will estimate these properties most quickly.
4. Experimentally, as the walk proceeds it naturally discovers high weight vertices, and the estimates based on these vertices are efficient after a suitable number of steps.

5. If the walk starts from a high weight vertex then, experimentally, the property estimates we obtain converge rapidly towards the correct value.

Our assumptions are that the network is unknown but can be queried through either a public API or via a web interface. In addition, we assume that all computations are based on the query results which are held locally on a single processor. The random walk runs on the query data and is used as a randomized algorithm to determine the next step in the query process. Our measure of computational complexity is the number of transitions made by the walk, and our aim is to obtain results in a number of steps sub-linear in the network size, which is assumed unknown.

There are four techniques for the design of algorithms to estimate graph properties based on random walks.

Cycle formula of regenerative processes (CFRP). This is based on obtaining a cumulative sum of a function when a RW first returns to the starting vertex. This has been used by Massoulié *et al.* [100], as an estimator of the number of vertices n, and in that paper the authors remark on its general applicability to property estimation.

Weighted Random Walk (WRW). This is based on obtaining the return time of a WRW to a vertex. This is a method we have developed for the purpose of estimating network properties.

Optimized Metropolis Hastings Random Walk (MHRW). This approach, used by [131], gives a weight A to vertices not having the desired property, and a weight B to those with the property. These values are incorporated into the stationary distribution and optimized.

Running totals. This is based on obtaining a cumulative sum of a function after an arbitrary number of RW steps S. This is a variation of the CFRP which is useful in estimating network averages.

The circumstances under which each technique is most appropriate are not well understood, and indeed the techniques can easily be combined. It may be that using the CFRP technique with a SRW is appropriate for structurally homogeneous graphs, and the WRW for structurally inhomogeneous graphs. The advantage of the CFRP technique is that it can used with a SRW, for which mixing times are often known. The advantage of a WRW is that is can exploit variations in graph structure. The advantage of the running totals method is that it does not rely on return times, which in some graphs may have a high variance. The underlying theory of the topic of weighted walks is underdeveloped, especially with regard to mixing times and variance of first return times. On the whole the ideas explored in this chapter worked well in practice, so there is encouragement for further study.

The simplest way to study a network is to inspect it completely using, e.g., breadth first search. Failing this, the another simple statistical method is to sample vertices *uar*, using the methodology described in Section 4.2. In practice for large networks such as the WWW or OLSNs, neither of these methods is feasible, but the network can still be queried by some limited form of crawling or interaction with the network through an API. Our assumption is that query results are held locally on a single processor. The selection of the next vertex to visit (query) is based on a a RW runs on the query data. The RW is used as a randomized algorithm to determine the next step in the query process. The measure of computational complexity is the number of steps made by the walk, and our aim is to have obtained results in a number of steps in the order of $O(cn)$ where $0 < c \leq 1$ and n is the network size, which is assumed unknown.

In this chapter we present the aforementioned methods to estimate graph properties. The chapter is organized as follows: In Section 5.1 we present the general methodology of the property estimation process. The discussion will include both a theoretical discussion, as well as the general procedure followed in order to obtain estimates. In Section 5.2 we will present details on the exact procedure followed to estimate each specific property. We will present details on obtaining estimates using each of the four methods described above, where applicable. In Section 6.1 we present experimental results obtained when

running our methods on theoretical graphs such as preferential attachment graphs. In addition we present results obtained using real large online network data which were freely available. We will also compare results on some estimators on real network data with configuration graphs of the same degree distribution.

5.1 Methodology

In this section we will present the general methodology which the property estimation methods we present are based on.

5.1.1 First return times of Random Walks.

In this section we describe the basis of the property estimators based on first return times of RWs. The methods for estimating network properties, presented here are based on using first return time of a random walk, given by Equation (3.10), i.e. $\mathbf{E}_v T_v^+ = \frac{1}{\pi_v}$. For a random walk starting from vertex v, with a first return time to v as defined in (3.10). If the walk is ergodic, it has a well defined stationary distribution π_v at any vertex v.

Assuming a random walk starting from vertex u at time $t = 0$ we can experimentally measure the first return time $T_u^+ = Z$. By repeating this process k times we obtain the sample $\widehat{Z} = \{Z_1, Z_2, \cdots, Z_k\}$. Let $Z(k) = \sum_{i=1}^{k} Z_i$, i.e., the cumulative time taken for k returns to vertex u. The random variable $Z(k)/k$ has expected value $\mathbf{E}T_u^+ = 1/\pi_u$, which for a WRW can be written as:

$$\mathbf{E}T_u^+ = \frac{1}{\pi_u} = \frac{w_G}{w(u)}, \tag{5.1}$$

where $w(u)$ is the weight of vertex u (i.e. the sum of the weights of the edges adjacent to u) and w_G is the weight of the graph (i.e. the sum of the weights of all vertices in the graph). This follows from the definition of the WRW. We now use Equation (5.1) as a property estimator. We assume that we know $w(u)$ for each vertex u we encounter during the walk. This is a realistic assumption since we need to compute it in order to

determine the next step of the walk. We use this knowledge and the estimate $Z(k)/k$ and solve for w_G.

Since the process is Markovian, the sample points of $\widehat{Z}$ are *i.i.d.* and the variance of $Z(k)$ is

$$\mathbf{Var}\ Z(k) = k\mathbf{Var}\ T_u^+, \tag{5.2}$$

Further analysis on the variance of the first return time can be found in Section 3.1.5 and specifically in Equation (3.22) repeated below:

$$\mathbf{Var}\ T_v^+ = \frac{2Z_{vv} + \pi_v - 1}{\pi_v^2} = \frac{C}{\pi_v^2} = C(\mathbf{E}T_v^+)^2$$

where

$$1 - \pi_v \leq C \leq 2/(1 - \lambda_2).$$

As we can see, vertices with larger stationary distribution have a lower value for $\mathbf{E}T_u^+$ and therefore a lower variance of return time. This would mean they should rapidly provide a sample whose average matches the true value.

Using the Chebychev inequality,

$$\mathbf{Pr}(|Z(k) - \mathbf{E}Z(k)| \geq \varepsilon) \leq \frac{\mathbf{Var}Z(k)}{\varepsilon^2}, \tag{5.3}$$

by setting $\varepsilon = \sqrt{\omega\mathbf{Var}Z(k)}$, dividing the bracketed term on the left hand side by k and using (5.2), we obtain

$$\mathbf{Pr}\left(\left|\frac{Z(k)}{k} - \mathbf{E}T_u^+\right| \geq \sqrt{\frac{\omega\mathbf{Var}\ T_u^+}{k}}\right) \leq \frac{1}{\omega}.$$

Using $\mathbf{Var}\ T_v^+ = C(\mathbf{E}T_v^+)^2$, from the previous section, assuming $C\omega$ is small compared to k, but ω is large in absolute value, we can write $\epsilon = \sqrt{C\omega/k}$, so that

$$\mathbf{Pr}\left(\left|\frac{Z(k)}{k} - \mathbf{E}T_u^+\right| \geq \epsilon\mathbf{E}T_u^+\right) \leq \frac{1}{\omega}, \tag{5.4}$$

This gives a crude measure of convergence. For rapidly mixing walks, C is small. If we assume that we start a SRW from a vertex u such that $d(u) = \Delta(G)$ then $T_u^+ =$

$\min_{v \in V} T_v^+$. If we assume that G is sparse, i.e. $m = cn$ and that $d(u) = n^\alpha$ with $0 < \alpha < 1$ then $\mathbf{E}_u T_u^+ = O(n^{1-\alpha})$ which is sub-linear in the size of the graph. This means, for a sufficiently small s we can collect s samples in (expected) sub-linear time $O(sn^{1-\alpha})$.

The concentration $\epsilon = \sqrt{C\omega/k}$, is based on the assumption that the walk is rapidly mixing (C constant) and that a large enough number, k, of return time samples can be taken to make ϵ small. In order to get k large, and to also keep the run time t of the walk short, we need a vertex u for which $\mathbf{E}T_u^+ = 1/\pi_u$ is small. That is, we need a vertex u such that $\pi_u \gg \sum_v \pi_v/n$; i.e. a vertex u which has a stationary distribution which is much greater than the average. While this is true for power-law degree distribution graphs and SRWs it is not always the case. The SRW based methods which are based on the first return times such as the cyclic formula for regenerative processes (Section 5.1.2) and the edge estimator based on the SRW (Section 5.2.1) would not work well on graphs with near uniform stationary distributions, such as d-regular graphs or Erdős-Rényi random graphs, unless we are prepared to wait. For graphs with stationary distribution $\pi_v \approx 1/n$ for all vertices, we would have to wait for $\Omega(kn)$ walk steps for a large enough sample, whereas we aim to sample in sub-linear time, if possible.

However, since we also use WRWs to estimate various properties, if there are high weight vertices and the graph is an expander then the weight bias ensures that we will quickly discover high weight vertices **whp**. The advantage of this approach is that, in the case we suspect that the underlying graph has a near uniform stationary distribution, but know that the vertices with the desired property are non-uniformly distributed within the graph, we can design a WRW in such a way that we give those vertices a significantly higher weight.

Using the above procedure and an appropriately designed random walk with a graph weight $w_G \propto T(G)$, where $T(G)$ is some global property of G (or the reciprocal of a property), we show that we can estimate $T(G)$ by estimating w_G. Examples of such estimators follow in Section 5.2.

The quantity $\mathbf{E}_\pi T_v$ can be bounded in various ways, to give estimates of Z_{vv} and $\mathbf{Var}\, T_v^+$. We give two methods: (M1) an estimate based on eigenvalue gap; (M2) a direct estimate from the return time data. One standard deviation of the sample mean for estimates of edges was derived by these methods. Illustration of these methods can be found in Chapter 6 along with all experimental results of these methods.

M1. From (3.13) we have

$$Z_{vv} = \sum_{t\geq 0}(P^{(t)}(v,v) - \pi_v) \leq \sum_{t\geq 0}|P^{(t)}(v,v) - \pi_v|$$

Using the result (see e.g. [95]) that $|P^{(t)}(v,v) - \pi_v| \leq \lambda_2^t$, gives

$$Z_{vv} \leq \frac{1}{1-\lambda_2}. \tag{5.5}$$

M2. We estimate Z_{vv} directly from the first return time data. Let τ_2 be a mixing time of a random walk on a graph G. The method described in [29] states (subject to certain technical conditions) that for $t > \tau_2$ the probability $\rho(t)$ that a first return to v has not occurred by t is of the form

$$\rho(t) \sim \exp\left(-t/\mathbf{E}_\pi T_v^+\right),$$

i.e.

$$\log\frac{1}{\rho(t)} = \frac{t}{\mathbf{E}_\pi T_v^+}.$$

Replacing $1-\rho(t)$ by the proportion $y(t)$ of returns at or before step t, estimates Z_{vv}.

When we change from a simple random walk S to a weighted walk W, we change the second eigenvalue $\lambda(W) = \lambda_2$ of the transition matrix, and hence the mixing rate. Some bounds on the change in eigenvalue gap can be obtained from the *Direct Comparison Lemma* for weighted random walks (Lemma 29, Ch. 3 [2]).

Lemma 5.1 (Direct Comparison Lemma [2])**.** *Let (w_e) and (w_e^*) be edge weights on a simple graph G, let (w_v) and (w_v^*) be vertex weights, and let τ_2, τ_2^* be the relaxation times for the associated random walks. Then*

$$\frac{\min_e(w_e/w_e^*)}{\max_v(w_v/w_v^*)} \leq \frac{\tau_2}{\tau_2^*} \leq \frac{\max_v(w_v/w_v^*)}{\min_e(w_e/w_e^*)}.$$

Put simply, Lemma 5.1 states that if the values of w_e and w_e^* as well as the values of w_v and w_v^* do not change by much, their ratio will remain close to 1 and therefore the mixing time $\tau_2^* \approx \tau_2$.

5.1.2 Cycle formula of regenerative processes (CFRP)

The CFRP can be summarized as follows. For a walk starting from vertex u at step 0, let X_t be the vertex occupied by the walk at step t. Let $f(X_i)$ be a function with the domain of f being V and with finite mean. Then

$$\mathbf{E}_u\left(\sum_{i=0}^{T_u^+-1} f(X_i)\right) = \mathbf{E}T_u^+ \sum_{v\in V} \pi_v f(v). \tag{5.6}$$

This identity is a consequence of Equation (3.16) by replacing S with T_u^+:

$$\mathbf{E}_u(\text{visits to } v \text{ before time } T_u^+) = \frac{\pi_v}{\pi_u} = \mathbf{E}T_u^+ \, \pi_v.$$

The identity (5.6) was used by Massoulié *et al.* [100] to count the network size using a simple random walk. Putting $f(v) = 1/d(v)$ removes the degree bias from π_v so that $\sum_{v\in V} \pi_v f(v) = n/2m$. The paper [100] uses the notation $f(v) = \phi(v)/d(v)$.

We maintain the convention $f(v) = \phi(v)/w(v)$ when generalizing to weighted random walks, with $\pi_v = w(v)/w_G$. Denote by $S(T_u^+)$ the random variable $\sum_{i=0}^{T_u^+-1} f(X_i)$, with expectation $\mathbf{E}S(T_u^+)$ given by (5.6). Let $\overline{\phi} = \sum_{v\in V} \phi(v)$, then, as $\mathbf{E}T_u^+ = 1/\pi_u$,

$$\begin{aligned}
\mathbf{E}S(T_u^+) &= \mathbf{E}T_u^+ \times \sum_{v\in V} \pi_v \frac{\phi_v}{w(v)} \\
&= \frac{w_G}{w(u)} \sum_{v\in V} \frac{w(v)}{w_G} \frac{\phi(v)}{w(v)} \\
&= \frac{1}{w(u)} \sum_{v\in V} \phi(v) \\
&= \frac{\overline{\phi}}{w(u)}
\end{aligned} \tag{5.7}$$

We can rewrite Equation 5.7 as:

$$\overline{\phi} = w(u)\mathbf{E}S(T_u^+) \tag{5.8}$$

which we use as an estimator for $\overline{\phi}$. If we set $\phi(u) = 1$ we have $\overline{\phi} = n$ and therefore we can use Equation (5.8) to estimate the number of vertices.

The variance of $S(T_u^+)$ for the case where $\phi(v) = 1$, was obtained in [100]. The variance $\mathbf{Var}\ \hat{n}$ mentioned in [100] is:

$$n^2\left[2\left(1-\frac{1}{n}\right)^2 - 1 - nd(u)\right] \leq \mathbf{Var}\ \hat{n} \leq n^2\frac{2d(u)}{1-\lambda_2}$$

where λ_2 is the second largest eigenvalue.

Since $w(u)$ is generally known, we can use the methodology described in Section 5.1.1, but this time we would measure $S(T_u^+)$ rather than T_u^+. The CFRP is a generalization of the first return times method, where if we set $f(u) = 1$ (i.e. $\phi(u) = w(u)$). we obtain the result: $\mathbf{E}S(T_u^+) = \frac{w_G}{w(u)} = \mathbf{E}T_u^+$ which is equivalent to Equation 5.1.

5.1.3 Method of running totals

A more general expression form related the cycle formula can be written as follows:

$$\mathbf{E}_\pi\left(\sum_{i=0}^{t} f(X_i)\right) = t\ \sum_{v\in V}\pi_v f(v). \tag{5.9}$$

where $\mathbf{E}_\pi$ denotes expectation from stationarity. We denote the number of visits to vertex u before time S by $N_u(S)$. We remind that a stationary Markov Chain is at vertex u with probability π_u, which in turn gives Equation (5.9) a similar interpretation to Equation (3.16) but from stationarity. This can be trivially determined by considering the fact that $\mathbf{E}_\pi N_u(t) = t\pi_u$.

Based on Equation 5.9, we generalize the estimator as follows:

$$\begin{aligned}\mathbf{E}_\pi S(t) &= t\sum_{v\in V}\pi_v\frac{\phi_v}{w(v)}\\ &= t\sum_{v\in V}\frac{w(v)}{w_G}\frac{\phi(v)}{w(v)}\\ &= \frac{t}{w_G}\sum_{v\in V}\phi(v)\\ &= \frac{t\overline{\phi}}{w_G}\end{aligned} \tag{5.10}$$

where $S(t)$ is a generalization of $S(T_u^+)$ such that $S(t) = \sum_{i=0}^{t-1} f(X_i)$.

This would allow us to estimate $\overline{\phi}$ as follows:

$$\overline{\phi} \approx \frac{S(t)w_G}{t}$$

The disadvantages of this method are:

- Knowledge of the mixing time τ_2 is required. In order to sample from stationarity we need to run a walk for τ_2 steps until it mixes, before we start sampling. While this is true for Equation (5.10), Dinwoodie [42], and Lezaud [91] have shown that, provided t is sufficiently large, results are accurate even when starting from any initial distribution.
- Knowledge of w_G is required in order to obtain estimates for $\overline{\phi}$. w_G is in many cases unknown.

However the method is useful in the cases where (a) we know w_G or have estimated it within a reasonable margin, or (b) we wish to obtain the network average values of ϕ , i.e. $\frac{1}{n}\overline{\phi}$. The idea of estimating network averages will be further explained in Section 5.2.3. We will now proceed to discuss previous work on the accuracy bounds for the running totals method both in the cases when assuming knowledge of the mixing time τ_2 and when we do not assume knowledge of τ_2.

Accuracy of the method of running totals

The following section concerns the empirical accuracy of the sampled running total $S(t)$ as given by (5.10). The total $S(t) = \sum_{i=1}^{t} f(X_i)$ depends on a function $f(X_i)$ evaluated at the vertices X_i visited by a random walk during steps $i = 1, ..., t$. Theorems 5.1–5.3 below provide Chernoff-type bounds for the concentration of the random variable $S(t)$. Theorem 5.1 seems the most straightforward in application, but requires the walk to start from the stationary distribution. The bounds in Theorems 5.2 and 5.3 allow any initial distribution of the random walk.

The following result from Léon and Perron [81] concerns sampling a function $f(v) : v \in V$ on a graph $G = (V, E)$ using a reversible ergodic random walk starting from stationarity. The function $f(v)$ evaluated at any vertex v is restricted to take values in the interval $[0, 1]$. Let π be the stationary distribution of the walk, and $\mu = \mathbf{E}_\pi f(X) = \sum_{v \in V} \pi_v f(v)$ the expected value of $f(X)$ w.r.t. the stationary distribution. Let $\lambda_0 = \max(\lambda_2, 0)$ where λ_2 is the second largest eigenvalue of the transition matrix P of the random walk. Let $\mathbf{Pr}_\pi$ denote the probability of an estimate for a walk starting from stationarity. Then the following theorem holds.

Theorem 5.1 (Léon and Perron [81])**.** *Consider an ergodic and reversible Markov Chain $(X_0, X_1, \ldots)$ on a graph $G = (V, E)$, starting from the stationary distribution π. Let $S(t) = \sum_{i=1}^{t} f(X_i)$ be the running total. Then, for any $\epsilon > 0$ such that $\mu + \epsilon < 1$,*

$$\mathbf{Pr}_\pi(S(t) \geq t(\mu + \epsilon)) \leq \exp\left(-2\frac{1 - \lambda_0}{1 + \lambda_0} t\epsilon^2\right). \tag{5.11}$$

An alternative approach used by Dinwoodie [42] and Lezaud [91] is to estimate such probabilities starting from a given initial distribution $q = (\mathbf{Pr}(X_0 = v : v \in V)$ on vertices. Theorems 5.2 and 5.3 below assume that the function f is mean centred, i.e., $\mu = \mathbf{E}_\pi f = \sum_{v \in V} \pi_v f(v) = 0$.

The following Chernoff-type bound for Markov Chains is from Lezaud [91].

Theorem 5.2 (Lezaud [91])**.** *Consider an ergodic and reversible Markov Chain $(X_0, X_1, \ldots)$ on a graph $G = (V, E)$. Let $S(t) = \sum f(X_i)$ be the running total. Assume $\max_v |f(v)| \leq 1$. Let λ_2 be the second largest eigenvalue of the transition matrix P of the walk. Then, for any initial distribution q of X_0, any positive integer t and all $0 \leq \gamma \leq 2/5$,*

$$\mathbf{Pr}_q\left(S(t) \geq t\gamma\right) \leq e^{(1-\lambda_2)/5} \cdot \sum_{u \in V} \frac{(\mathbf{Pr}(X_0 = u))^2}{\pi(u)} \cdot \exp\left(-\frac{t\gamma^2(1 - \lambda_2)}{4}\left(1 - \frac{5\gamma}{2}\right)\right).$$

A recent paper by Wagner [127] gives simplified bounds for Lezaud type inequalities given the variance $\sigma^2 = \sum_{v \in V} \pi_v f^2(v)$ of f. An example of this is:

Theorem 5.3 (Wagner [127])**.** *Consider an ergodic and reversible Markov Chain $(X_0, X_1, \ldots)$ on a graph $G = (V, E)$. Let $S(t) = \sum f(X_i)$ be the running total. Let $\lambda = \max(\lambda_2, 0)$, where λ_2 is the second eigenvalue of the transition matrix P of the walk.*

Assume $\max_v |f(v)| \leq 1$. *Then, for any initial distribution* q *of* X_0, *any positive integer* t *and all* $\gamma > 0$,

$$\mathbf{Pr}_q\left(S(t) \geq t\gamma\right) \leq e^{\frac{1-\lambda}{1+\lambda}\frac{\gamma^2}{\sigma^2+\gamma}} \cdot \sum_{u \in V} \frac{(\mathbf{Pr}(X_0 = u))^2}{\pi(u)} \cdot \exp\left(-\frac{t}{8\sigma^2}(1-\lambda)\gamma^2\right).$$

The following result from K. Chung *et al.* [22] concerns general Markov Chains.

Theorem 5.4 (Chung *et al.* [22])**.** *Assume an ergodic Markov Chain over state space* V *and stationary distribution* π. *Let* $T = T(\epsilon)$ *be its* ϵ*-mixing time for* $\epsilon \leq \frac{1}{8}$ *and let* $(X_0, \ldots, X_t)$ *denote a* t *step random walk starting from initial distribution* q *(i.e.* X_0 *is drawn from distribution* q*). Assume* $f_i \to [0,1]$ *with* $\mathbf{E}f_i = \mu$ *and* $S(t) = \sum_{i=0}^{t} f_i(X_i)$. *There exists a constant* c *such that:*

$$Pr_q(|S(t) - \mu t| \geq \delta\mu t) \leq c\sqrt{\sum_{i=1}^{n} \frac{q_i^2}{\pi_i}} \exp\left(\frac{-\delta^2\mu t}{72T}\right) \qquad \text{for } 0 \leq \delta \leq 1 \tag{5.12}$$

$$Pr_q(S(t) \geq (1+\delta)\mu t) \leq c\sqrt{\sum_{i=1}^{n} \frac{q_i^2}{\pi_i}} \exp\left(\frac{-\delta\mu t}{72T}\right) \qquad \text{for } \delta > 1 \tag{5.13}$$

Overall, the error in the estimate of the running totals method decreases exponentially with the sample size. There are a few things to note on Theorems 5.1-5.4.

- Theorems 5.1-5.3 provide an upper bound on the error of the estimates **whp**. We generally don't know the probability $P(|S(t) - t\mu| < t\epsilon)$ in these cases. This means that while we can use the results of Theorems 5.1-5.3 to determine a **whp** upper bound on any estimates, we cannot use them to determine a lower bound. This is not the case for Theorem 5.4.
- Theorem 5.1 and Theorem 5.4 hold for $\mu < 1$ while Theorems 5.2 and Theorem 5.3 hold for $\mu = 0$. This means that in order to be able to use these results, we may need to rescale the function $f(u)$ appropriately.

5.2 Graph property estimators

In this section we present some property estimators based on the four methods we have described in the beginning of this chapter.

5.2.1 Edge estimator

Based on the return time of a Simple Random Walk

A simple example of our approach, is to estimate the number of edges m using a RW. As seen in based on the discussion in Section 3.3, setting an edge weight $w(u, v) = 1$ leads to $w(u) = d(u)$ and $w_G = 2m$. Let $Z(k) = \sum_{i=1}^{k} Z_i$ be time for the k-th return to vertex u. We can use the random variable

$$\widehat{m} = \frac{Z(k)d(u)}{2k} \tag{5.14}$$

to estimate the total number of edges m of a graph without the need to exhaustively count them.

Edge Estimator using the CFRP

If we wish to use the CFRP to estimate the number of edges the procedure is as follows we set a function $\phi(u) = d(u)$. It is clear that in the case we use a SRW, function $f(u) = \frac{\phi(u)}{w(u)}$ becomes $f(u) = 1 \quad \forall u \in V$. This is essentially equivalent to counting the steps of first return which our previous estimator was based on. However, in this case we may choose to use a WRW as the underlying walk if for any reason a SRW is not a viable option.

Edge Estimator Based on collision sampling

A SRW in stationarity, is uniform among the edges which can be easily determined by considering the probability of transitioning along any edge $e = \{u, v\}$ during step t is the probability of being at vertex u at step $t - 1$ and then selecting vertex v or (since the walk is reversible) vice-versa. This is:

$$\begin{aligned} Pr(X_e) &= \pi_u P_{uv} + \pi_v P_{vu} \\ &= \frac{\pi_u}{d(u)} + \frac{\pi_v}{d(v)} \\ &= \frac{1}{m} \end{aligned}$$

This means that we can use the sample and collide method as described in Section 4.1.5. We remind that based on the sample-and-collide method, the expected number of samples before the first collision should be $\sqrt{2m}$ (since the set we are sampling from is the set of edges). In general given a sample size R and a number of collisions C the estimated number of edges is:

$$\widehat{m} \approx \frac{R^2}{2C} \tag{5.15}$$

In practice in order to deploy this method and estimate the number of edges we either need to know the mixing time of the walk or make a "good guess".

5.2.2 Random Walk based vertex estimators

As we have noted in Chapter 4, the size of a population is an important property and multiple methods exist to estimate it. We have described some of these methods which were mainly based on uniform sampling. However since, as we mentioned, we assume that sampling *uar* directly is not possible in the case of OLSNs and WWW, therefore we have also presented some methods to obtain *uar* samples by unbiasing RWs.

In this section we extend the discussion of Sections 4.1.5 and 4.2. We present two methods to estimate the number of vertices, the first method uses a sample-and-collide method with non uniform samples obtained through a random walk, in order to estimate the number of vertices. We then present a method we have devised to estimate the number of vertices. This method is based on an a WRW with appropriately assigned weights which allows us to use the method of first returns to estimate the number of vertices.

Sample-and-Collide Based Vertex estimator using biased sampling

As pointed out by L. Katzir *et al.* [71] and discussed in Section 4.1.5, it is possible to estimate the number of members of a group (i.e., vertices in a graph) by counting the number of collisions that an unbiased sampling would yield. In [71] this notion is further

extended to yield similar results using a SRW on a directed graph. While in our case the graphs in question are not directed the principles remain the same.

Assume a sample of r vertices $\{x_1, x_2, \cdots, x_r\}$ taken from a sampling distribution π the stationary distribution of a SRW. Define $C = \sum_{i=1}^{r} \sum_{j=i+1}^{r} I(x_i = x_j)$ where I is the indicator function taking the value of 1 if the predicate within $(x_i = x_j)$ is satisfied, and 0 otherwise. If we assume stationarity then it is reasonable that the samples x_i are independent, since samples from stationarity are not correlated with the starting vertex or any other vertex and are always drawn from the stationary distribution.

The expectation $\mathbf{E} = \mathbf{E}_\pi$ of a vertex dependent random variable X sampled according to the stationary distribution $\boldsymbol{\pi}$ of a random walk is:

$$\mathbf{E}_\pi X = \sum_{v \in V} X(v)\pi_v.$$

The expected number of collisions is:

$$\begin{aligned} \mathbf{E}C &= \mathbf{E} \sum_{i=1}^{r} \sum_{j=i+1}^{r} I(x_i = x_j) \\ &= \binom{r}{2} \mathbf{E} I(x_i = x_j) \\ &= \binom{r}{2} \sum_{v \in V} \pi_v^2 \end{aligned} \tag{5.16}$$

The expected sum of the degrees of the sample is:

$$\mathbf{E}\hat{m} = \mathbf{E} \sum_{i=1}^{r} d(x_i) = r \sum_{v \in V} d(v)\pi_v = \frac{r}{2m} \sum_{v \in V} d(v)^2. \tag{5.17}$$

Finally the expected reciprocal sum of the degrees is:

$$\mathbf{E}\hat{m}^{-1} = \mathbf{E} \sum_{i=1}^{r} d(x_i)^{-1} = r \sum_{v \in V} \frac{\pi_v}{d(v)} = \frac{rn}{2m}$$

If the term n is isolated from the above then the vertex estimator is as follows:

$$\hat{n} = \binom{r}{2} \frac{\mathbf{E}\hat{m}\mathbf{E}\hat{m}^{-1}}{r^2 \mathbf{E}C} = \frac{r-1}{r} \frac{\mathbf{E}\hat{m}\mathbf{E}\hat{m}^{-1}}{2\mathbf{E}C} \tag{5.18}$$

Vertex estimator based on first return times

We use an inversely degree biased WRW. Let $\gamma > 0$ be some positive constant. We set the edge weight $w(u,v) = \frac{\gamma}{d(u)} + \frac{\gamma}{d(v)}$. Let VRW be the associated walk, then the stationary distribution of the walk is:

$$\pi_u = \frac{w(u)}{w_G} = \frac{\gamma + \sum_{v \in N(u)} \frac{\gamma}{d(v)}}{2\gamma n}. \tag{5.19}$$

This follows as

$$\begin{aligned} w(u) &= \sum_{v \in N(u)} w(u,v) \\ &= \sum_{v \in N(u)} \frac{\gamma}{d(u)} + \sum_{v \in N(u)} \frac{\gamma}{d(v)} \\ &= \gamma + \sum_{v \in N(u)} \frac{\gamma}{d(v)} \end{aligned} \tag{5.20}$$

$$\begin{aligned} w_G &= 2\sum_{e \in E} w(e) \\ &= 2\sum_{e \in E} \frac{\gamma}{d(u)} + \frac{\gamma}{d(v)} \quad = \quad 4\sum_{e \in E} \frac{\gamma}{d(u)} \\ &= 2\sum_{u \in V} \sum_{v \in N(u)} \frac{\gamma}{d(u)} \quad = \quad 2\gamma n. \end{aligned}$$

Let $Z(k) = \sum_{i=1}^{k} Z_i$ be time for the k-th return to vertex u. We use the following estimator for vertices:

$$\widehat{n} = \frac{Z(k)w(u)}{2\gamma k} \tag{5.21}$$

where $w(u)$ is given by Equation 5.20.

5.2.3 Triangle estimator

In this section we will present a few methods to estimate the number of triangles. In general number of triangles is an interesting property since a higher than random density of triangles is generally viewed as an indicative property of a OLSN.

The methods presented here have a significant overhead introduced at each step, since we generally require knowledge of $t(u)$, the number of triangles which include vertex u. For each vertex u this requires $O(d(u)^2)$ time. However since exhaustively counting the number of triangles is itself a hard problem, with the complexity of the current "state of the art" algorithm being $O(n^\omega)$ where ω is the fast matrix multiplication exponent[1]. In addition to $O(n^\omega)$ being very high for very large graphs, this specific algorithm requires the graph to be represented as an adjacency matrix, requiring $\Theta(n^2)$ space. While there are space optimizations for sparse matrices, the run time is generally the same.

WRW based triangle estimator

For each edge e we assign a weight $1 + ct(e)$ where $t(e)$ is the number of triangles containing e. Let $t(v)$ be the number of triangles containing v, and $t(G)$ the total number of triangles in G and $c \geq 1$. Let TRW be the associated random walk, then

$$\pi_u = \frac{w(u)}{w_G} = \frac{d(u) + 2ct(u)}{2m + 6ct(G)}.$$

This follows as

$$\begin{aligned} w(u) &= \sum_{v \in N(u)} 1 + ct(e) \\ &= d(u) + 2ct(u) \\ w_G &= \sum_{u \in V} w(u) \\ &= \sum_{u \in V} d(u) + 2ct(u) \\ &= 2m + 6ct(G) \end{aligned}$$

Let $Z(k) = \sum_{i=1}^{k} Z_i$ be time for the k-th return to vertex u. We can use this to estimate the number of triangles $t(G)$ by

$$\widehat{t} = \frac{Z(k)(d(u) + 2ct(u)) - 2m}{6ck}, \tag{5.22}$$

[1]At the time of writing $\omega = 2.3729$, by Williams [130]. The algorithm which achieves runtime $O(n^{2.3729})$ has large constants hidden in the Big-O notation meaning that in practice, for graphs which can be stored in todays memory, the runtime of $O(n^{2.3729})$ is unachievable.

where m can be estimated by Equation (5.14). We note that for the majority of our experiments we have chosen to use $c = 1$. Ideally we would choose c such that $2m \approx 6ct(G)$ in order to equally distribute any errors in the estimate, to both terms of the weight.

Triangle Estimator using the CFRP

Using the method described in Section 5.1.2 we can create an estimator of the number of triangles by selecting $\phi(u) = t(u)$ meaning that $f(u) = \frac{t(u)}{w(u)}$ This results in:

$$\mathbf{E}S(T_u^+) = \frac{3t(G)}{w(u)}$$

We remind that $\mathbf{E}S(T_u^+)$ is the $\mathbf{E}\sum_{i=0}^{T_u^+-1} f(X_i)$. We can now make use of any random walk to estimate:

$$\widehat{t} = \frac{w(u)S(T_u^+)}{3} \tag{5.23}$$

5.2.4 Estimating the number of occurrences of an arbitrary fixed subgraph.

Using a weighted random walk to estimate the number of edges m or triangles $t(G)$ in a graph G are special cases of the following problem. Let $\mathcal{S}$ be a set of unlabelled graphs. For each $M \in \mathcal{S}$ we want to count the number of distinct labelled copies of M in the graph G. The cases edges and triangles given above correspond to $\mathcal{S} = \{K_2\}$ and $\mathcal{S} = \{K_2, K_3\}$ respectively. For each $e \in E(G)$ we put $w(e) = \sum_{M\in\mathcal{S}} N(M, e)$, where $N(M, e)$ is the number of distinct subgraphs H isomorphic to M which contain e. M can be any connected subgraph, e.g., $K_k, K_{k,\ell}$, a chordless cycle of length 4, some specific tree. The simplest case (after $\mathcal{S} = \{K_2\}$) is $\mathcal{S} = \{K_2, M\}$. In this case we have the following:

$$\begin{aligned} w_G &= 2\sum_{e\in E} w(e) = 2\sum_{e\in E}(1 + N(M, e)) \\ &= 2m + 2\nu\mu(G), \end{aligned} \tag{5.24}$$

where $\nu = |E(M)|$ and $\mu(G)$ is the number of distinct copies of M in G. As $\pi_v = w(v)/w_G$, and $w(v)$ and ν are known, we can use the method of first returns to estimate $\mu(G)$.

Estimating the number of occurrences of an arbitrary fixed subgraph using the CFRP.

Following the same ideas as above, we use the vertex valued function $f(u) = \frac{N(M,u)}{w(u)}$ where $N(M,u)$ is the number of subgraphs isomorphic to M which vertex u belongs to.

By using this function we have:

$$\mathbf{E}S(T_u^+) = \frac{\xi\mu(G)}{w(u)}$$

where $\xi = |V(M)|$ the number of vertices belonging to graph M.

This would mean that we can estimate $\mu(G)$ using the following equation:

$$\widehat{\mu}(G) = \frac{S(T_u^+)w(u)}{\xi} \tag{5.25}$$

We can observe that Equation (5.25) generalized the CFRP formulae seen so far, i.e. for the case of vertices $N(M,u) = 1$ for each vertex and $\xi = 1$, or for the case of triangles $N(M,u) = t(u)$ and $\xi = 3$.

5.2.5 Estimating average clustering coefficient

The average clustering coefficient is a network property which we discussed in detail in Chapter 2 and specifically in Section 2.1.8. We remind that the clustering coefficient of a vertex is given by Equation (2.13).

$$C_u = \begin{cases} \frac{2|\{e_{vw}\}|}{d(u)(d(u)-1)} & \text{if } d(u) > 1 \\ 0 & \text{otherwise.} \end{cases}$$

If $d(u) > 1$ we can write $C_u = \frac{2t(u)}{d(u)(d(u)-1)}$.

Let $\epsilon > 0, \gamma > 0$ be small positive constants, we can use the WRW with weights $w(u,v) = \epsilon + \gamma\frac{C_u}{d(u)} + \gamma\frac{C_v}{d(v)}$ to estimate the average clustering coefficient $C = \frac{\sum_{u\in V} C_u}{n}$. We note

that, as in the case of the TRW, the ϵ is added to the weight to ensure that the edge would still be traversable even if $C_u = C_v = 0$ and therefore the graph would remain connected. This is because:

$$\begin{aligned} w(u) &= \sum_{v \in N(u)} w(u,v) \\ &= \sum_{v \in N(u)} (\epsilon + \gamma \frac{C_u}{d(u)} + \gamma \frac{C_v}{d(v)}) \\ &= \epsilon d(u) + \gamma C_u + \sum_{v \in N(u)} \gamma \frac{C_v}{d(v)} \end{aligned}$$

and the graph weight is:

$$\begin{aligned} w_G &= 2 \sum_{(u,v) \in E} w(u,v) \\ &= 2\epsilon m + 2(\sum_{(u,v) \in E} \gamma \frac{C_u}{d(u)} + \sum_{(u,v) \in E} \gamma \frac{C_v}{d(v)}) \\ &= 2\epsilon m + 4 \sum_{(u,v) \in E} \gamma \frac{C_u}{d(u)} \\ &= 2\epsilon m + 2 \sum_{u \in V} \sum_{v \in N(u)} \gamma \frac{C_u}{d(u)} \\ &= 2\epsilon m + 2 \sum_{u \in V} \gamma C_u \\ &= 2\epsilon m + 2n\gamma C \end{aligned}$$

We remind that from the discussion in Section 2.1.8, we only consider this method on simple graphs.

The average clustering coefficient in this case can be estimated using:

$$\widehat{C} = \frac{Z(k)w(u) - 2\epsilon m}{2kn\gamma} \tag{5.26}$$

In this case we note a few significant drawbacks of this approach:

- The term $2m$ is much greater than C since $0 \leq C \leq 1$ and $m = O(n)$ in sparse graphs. The parameters ϵ and γ can be used to appropriately make the two terms

comparable, but it is not realistic to assume that we know appropriate values of these parameters without assuming we know the values of C and m.

- This method requires knowledge of m and n, which we assume as unknown. This assumption is similar to the assumption made when estimating the number of triangles or arbitrary subgraphs. The difference is that in practice we would have to use estimates of these values, which introduces an error. In the case of triangle counting, the error introduced only involves the term m but in this case it would involve both m and n.

In summary, the problem is that C is very small compared to w_G and that it is a value which is an average over the entire graph. By contrast, all previous properties estimated had values are typically much larger, and were summed over the entire graph.

While a WRW is not the most appropriate approach to estimate C, we will show how the method of the running totals can be used to greatly simplify the estimation of C, and any similar property.

Estimating average clustering coefficient using the method of running totals

We remind that the method of running totals is a method based on a generalization of the CFRP. We can use this method to estimate C by assuming a vertex valued function $f(u) = \frac{C_u}{w(u)}$. From Equation (5.10) we have:

$$\mathbf{E}S(t) = \frac{\sum_{v \in V} C_u}{w_G} = \frac{nC}{w_G}$$

by using the VRW with $w_G = 2\gamma n$ as the underlying walk, and setting $\gamma = \frac{1}{2}$ this is simplified to:

$$\mathbf{E}S(t) = C$$

and therefore $\widehat{C} = S(t)$.

This method is most appropriate since $0 \leq C_u \leq 1$ and in general $C < 1$. This would mean that we can use the results from Theorems 5.1,5.2,5.3 and 5.4 to obtain confidence

intervals. In practice what this means is that from, e.g., Theorem 5.1 we would have:

$$P(S(t) \geq t(C+\epsilon)) = P(\widehat{C} \geq C+\epsilon) \leq \exp\left(-2\frac{1-\lambda_0}{1+\lambda_0}t\epsilon^2\right)$$

5.2.6 Estimating network averages

In general, we can use the running totals of the CFRP in order to estimate any network average. Assume a vertex property with value $P(u)$. Our goal is to estimate the average value of $P(u)$ i.e. $\overline{P} = \frac{1}{n}\sum_{v\in V} P(u)$. If we set a vertex valued function $f(u) = \frac{P(u)}{w(u)}$ and by using the VRW with $\gamma = \frac{1}{2}$ we have:

$$\mathbf{E}S(t) = \frac{\sum_{v\in V} P(u)}{w_G} = \frac{n\overline{P}}{w_G} = \overline{P}$$

This idea we present here is based on the average clustering coefficient estimation, but can easily be extended to average degree $\overline{d}$ or degree correlations $\langle k_{nn}\rangle$.

5.3 Conclusion

In this section we have presented a variety of methods to estimate properties using RWs. We discussed three basic RW based methods in depth:

- Methods based on the first return times of RWs.
- Methods based on the CFRP.
- Methods based on running totals of a RW.

We have also presented methods to estimate properties such as: the number of edges m, the number of vertices n, the number of triangles t, the number of arbitrary fixed subgraphs $\mu(G)$ and the average clustering coefficient C. We presented ways we can use either of the aforementioned RW-based methods in order to estimate each of these properties.

Chapter 6

Estimating Network Properties: Experimental Results

In this chapter we will present experimental results of the property estimation methods discussed in Chapter 5. Most of the results which appear here are joint work with C. Cooper and T. Radzik and appear in [35, 38, 39]. The results here were obtained using a home computer with limited capabilities. The specifications of the machine were Intel Core 2 Quad with 8 GB RAM, which are slightly lower than the standard budget personal computers in circulation today. This shows, that in practice these methods do not require many resources in terms of computational power or memory. The simulations of the RWs were written in C#[1].

6.1 Sampling approach, and presentation of results.

The total number of steps of the walk in all experiments was, in most cases, limited to the graph size n. The results presented are averaged over 10 runs of the experiment. The following was recorded at each step of the walk for later processing: Step, Vertex identifier, Vertex weight, Cycle Formula function cumulate. Random Walk simulations rely on a good random number generation and therefore, in order to generate random numbers we used a random number generator developed by M. Matsumoto and T.

[1]The code of the simulation can be found at https://github.com/apokryfos/randomwalkframework

Nishimura [102] called the Mersenne twister random number generator the code we used was ported from a C code[2] by Rory Plaire.[3]

We present our findings in two different ways:

1. The property estimate for the highest-weight vertex the walk discovered, showing how this estimate changed with the number of returns to that vertex. We refer to this as *the high-weight property estimate.* Our hypothesis is that high weight vertices make good property estimators. The purpose of plotting by returns is to show convergence over time of the property estimate to its correct value (the horizontal line) for the highest-weight vertex.

2. The property estimate obtained for all sampled vertices at a fixed time s, showing how the estimate varies as a function of vertex weight. We refer to this as *the property estimate per vertex weight.* The purpose of plotting by weight is to show the long run convergence of the property estimate as a function of vertex weight.

The precise data presented is as follows.

For the highest weight property estimate. We take the highest weight vertex v which occurred in all runs and plot the evolution of the property estimate p as a function of the number of returns i. For the given vertex v, let $N(v, j)$ be the total number of returns to v in runs $j = 1, ..., 10$ of the simulation. The value $p_i = \frac{1}{10}\sum_{j=1}^{10} P(i, j)$ where $P(i, j)$ is the property estimate at return i in experiment j.

In addition, the actual estimates made for each individual experiment as displayed as points on the same plot. The variance of the estimate is also plotted as an envelope around the mean estimate to show convergence.

[2]The original C code can be found at `http://www.math.keio.ac.jp/~matumoto/mt.html`

[3]The C# code can be found at `https://github.com/pwhe23/Pathfinder/blob/master/Pathfinder.Domain/Utility/MersenneTwister.cs`

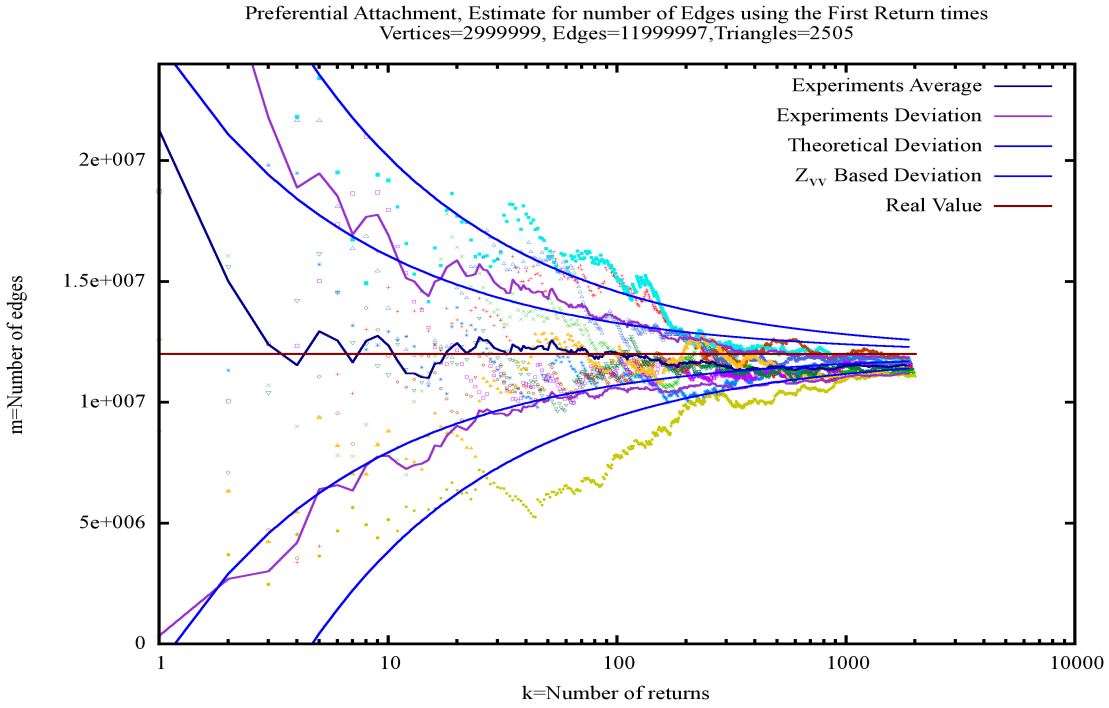
For the property estimate per vertex weight. The plot values are $(d, p(d))$, where d is weight and p property estimate. Let $n_j = N(d, s, j)$ be the number of vertices of weight d observed at step s in experiment j and let $P(d, s, j)$ be the average value of the property based on a sample of size n_j, then $p(d) = \frac{1}{10}\sum_{j=1}^{10} P(d, s, j)$. The results are given for steps $s = 0.3n, 0.6n, n$, and are plotted at three different levels. This makes it easy to compare the estimates at different steps, but also to distinguish them. The lowest level shows the estimates obtained when the number of steps is $s = 0.3n$. The level above that, shows the estimates when the number of steps is $s = 0.6n$, with the reference line for the property shifted upwards to twice its value. The third level shows the estimates when the number of steps is $s = n$, with the reference line shifted to four times its value.

6.1.1 Preferential Attachment Model

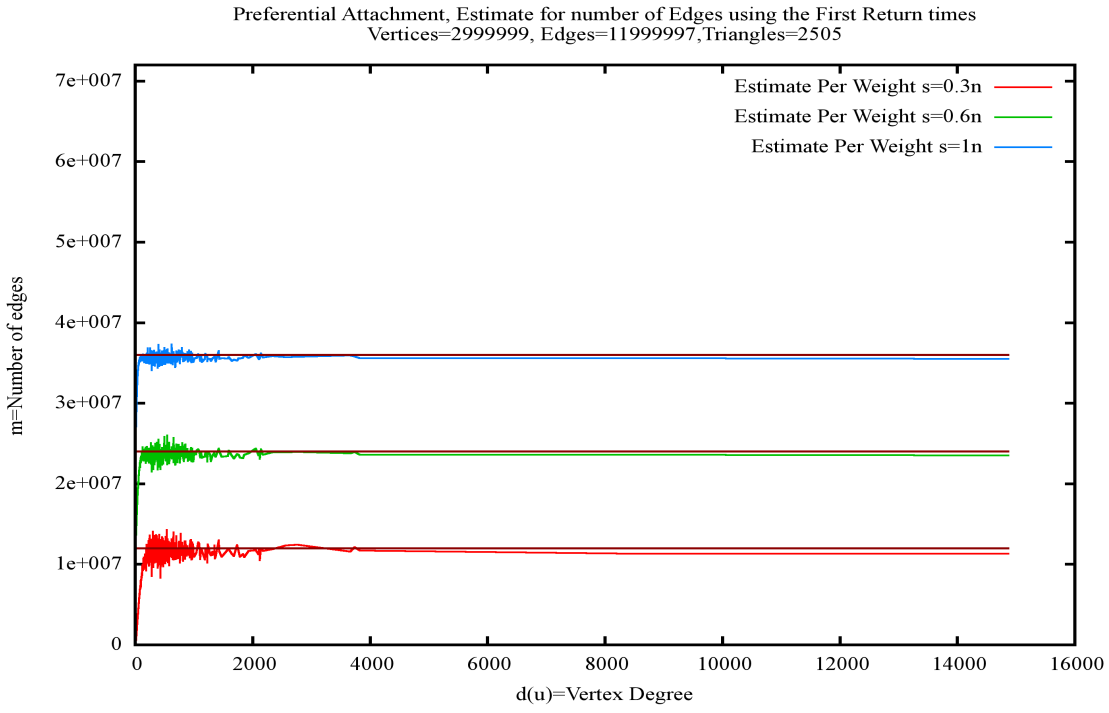
The preferential attachment graph used for these experiments is the graph model discussed in Section 2.2.2, specifically, the A. Barabási and R. Albert [4] where at each step, a new vertex joins the graph and attaches r edges preferentially to existing vertices.

We generated a graph with $3 \cdot 10^6$ vertices and $1.2 \cdot 10^7$ edges. Four edges were added at each step ($r = 4$) using preferential attachment. As the expected number of triangles in this model is small (about 5000 in the generated graph), no estimate was made of this quantity.

Estimate of number of vertices. In Figure 6.2a we can see the vertex estimate $\widehat{n}$ obtained using the walk described in Equation (5.21). Additionally Figure 6.2b shows the vertex estimate averaged for each weight. The conclusions are: (i) That the estimate based on the highest weight vertex visited by the walk converge rapidly (at around 50,000 steps), (ii) That the estimates do not change significantly with vertex weight after $0.3n$ steps have elapsed, (iii) Any vertex of reasonably high weight can be used as an estimator. Indeed above weight about 300, the estimates obtained do not differ significantly. This Figure 6.2a also indicates that initially the variation of results between experiments is

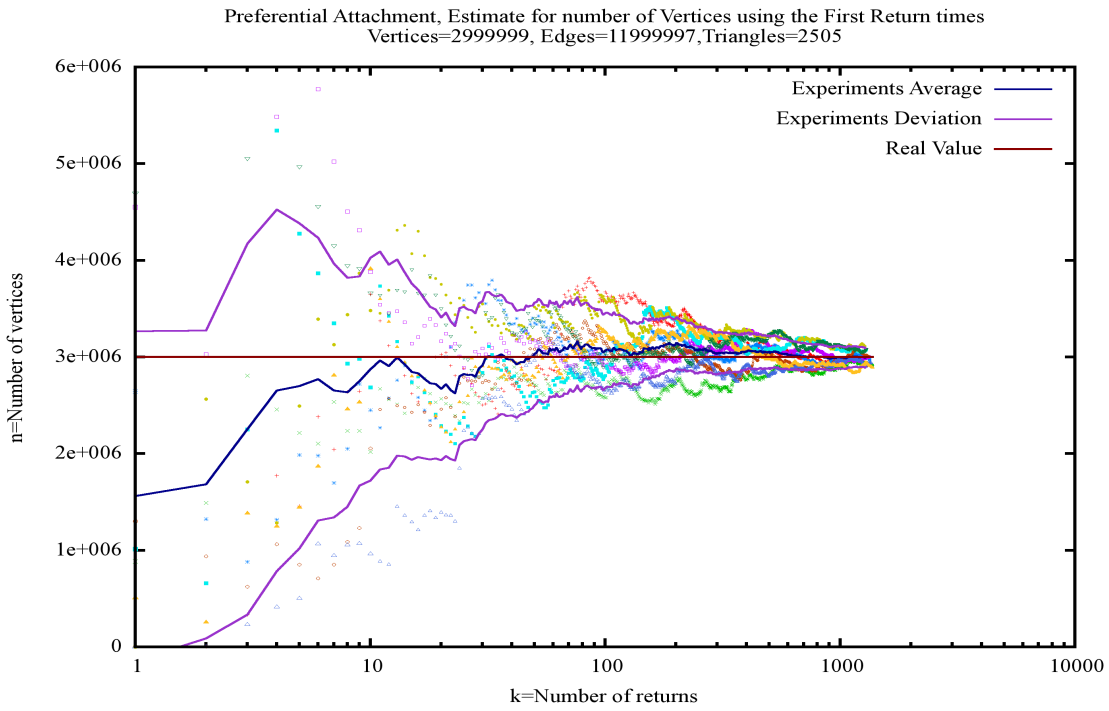


(a) Estimate based on returns to a high degree vertex

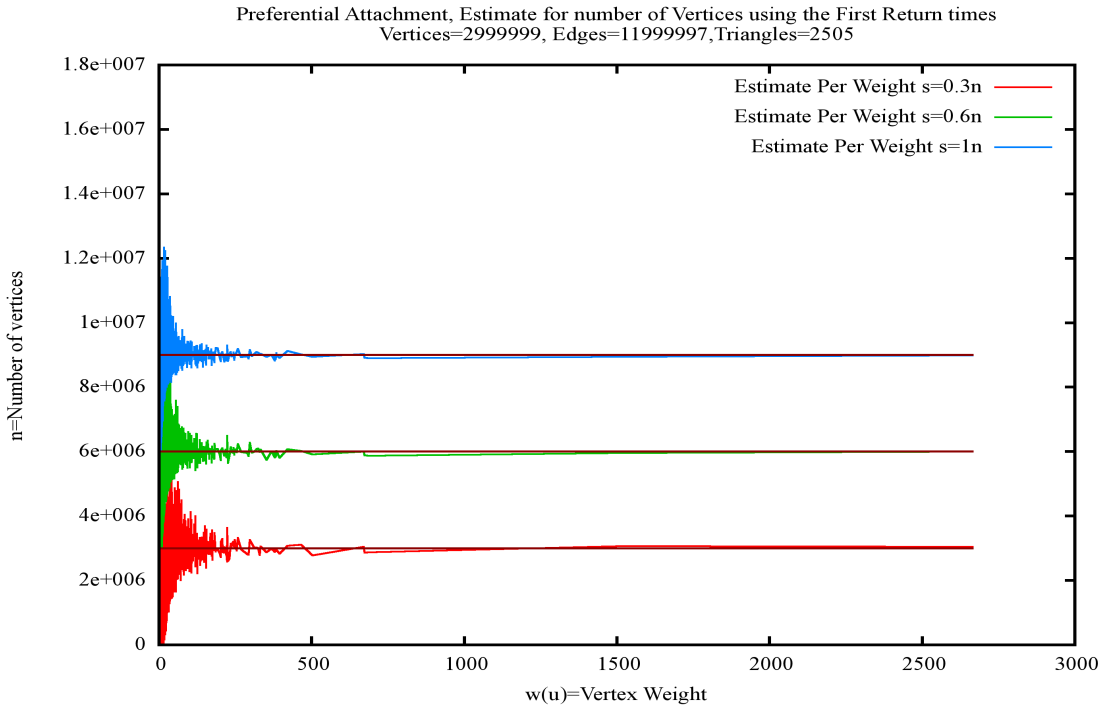


(b) Edges estimate as a function of vertex degree

Figure 6.1: Edge estimates for the preferential attachment graph based on the first return times of a SRW



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.2: Vertex estimates for the preferential attachment model based on the first return times of a VRW

very high, but the results for all experiments soon become similar, and an averaging approach can be justified.

An alternative way to estimate number of vertices, using the Cycle formula for regenerative processes (see Section 5.1.2) is also shown in Figures 6.3a and 6.3b. We note the accuracy of estimates using the Cycle formula is heavily correlated with the accuracy of estimating the graph weight using the corresponding SRW.

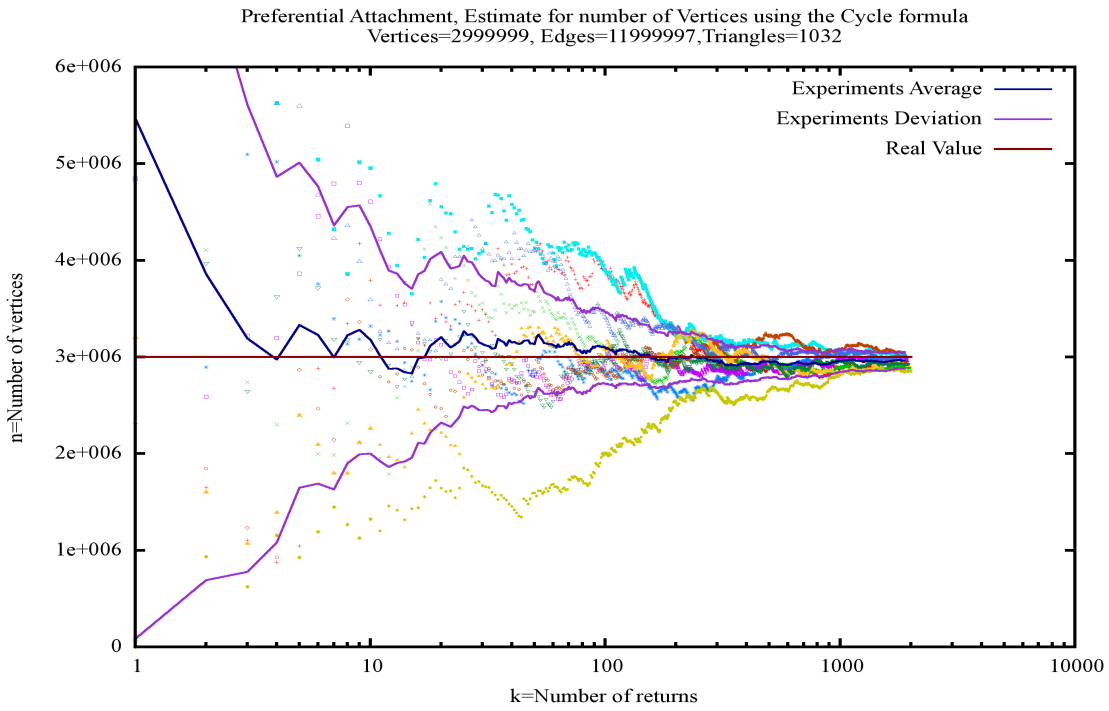
Estimate of number of edges. In Figure 6.1a we can see the edge estimate $\widehat{m}$ for this graph using a SRW. What is also seen in this figure is the theoretical variance of the estimates based on $\mathbf{Var}_\pi\, T_u^+$. The methods used to obtain this variance were discussed in Section 3.1.5. We remind that there were two methods described: Method M1 which is based on the eigenvalue gap $1 - \lambda_2$, and method M2 which is based on an estimate of the value of Z_{vv} based on return time data. The outer blue curve is obtained using method (M1) and the inner curve using method (M2) . For the plot shown in Figure 6.1a the estimate of $\widehat{Z}_{vv} = 1.6$ was obtained using the method M2.

In Figure 6.1b we can see the value of the edge estimate for all sampled vertices, averaged over the vertices of the same weight. The conclusions are similar to those of the vertex estimators, that experimentally, the estimation process is stable.

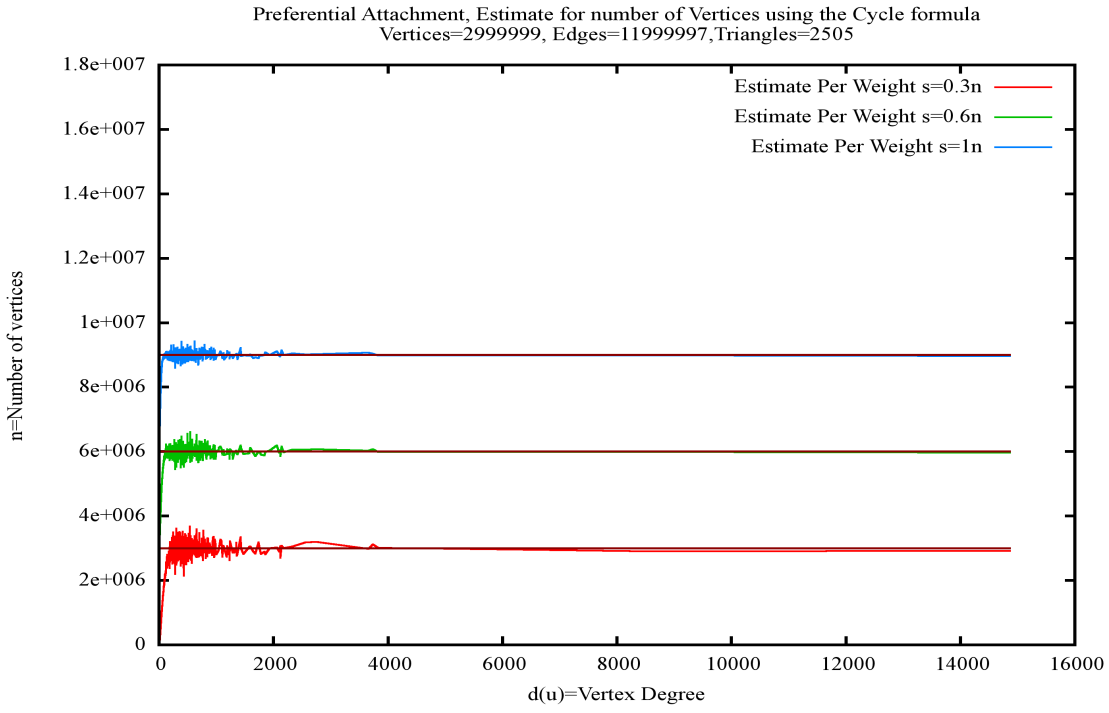
Importance of high degree/weight vertices. As we can observe in Figures 6.1b and 6.2b, vertices of higher degree (resp. weight) seem to converge quicker. This confirms our assertion that vertices which higher weights/degrees are important vertices for estimating properties. This agrees with what Equations 3.10 and 3.21 would suggest.

Edge estimates using edge collisions We also present the estimate for the number of edges using the birthday paradox and on Equation (5.15) we have attempted to estimate the number of edges using various values for our estimate of mixing time T. In detail, at each T-th step we sample the edge which we have traversed at that step. Having obtained R samples in this way we have created an estimate for the number of edges.

In Figure 6.4 we can see the estimates obtained in this way, for the preferential attachment graph mentioned above. We expect by setting T above $\log n \approx 15$ to obtain an



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.3: Vertex estimates for the preferential attachment model based on the CFRP

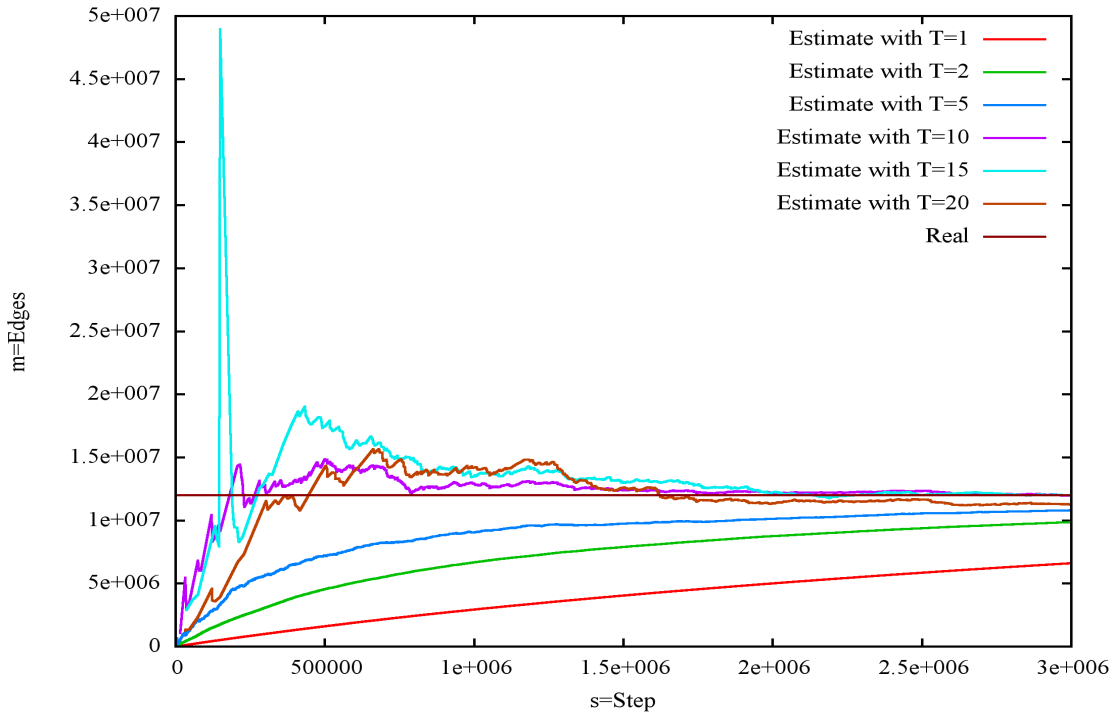


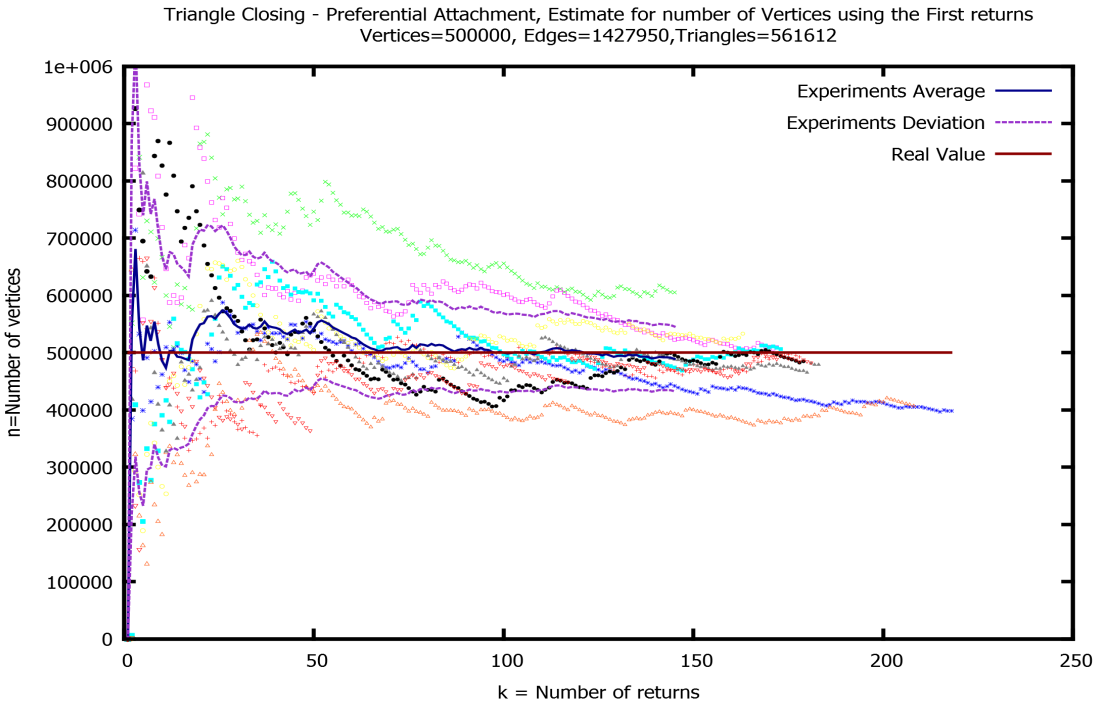
Figure 6.4: Edge estimate based on edge collisions on PA

accurate estimate. This is due to the fact that the mixing time τ_2 of a preferential attachment graph is noted to be $\tau_{2,PA} = O(\log n)$ (see e.g. [25]).

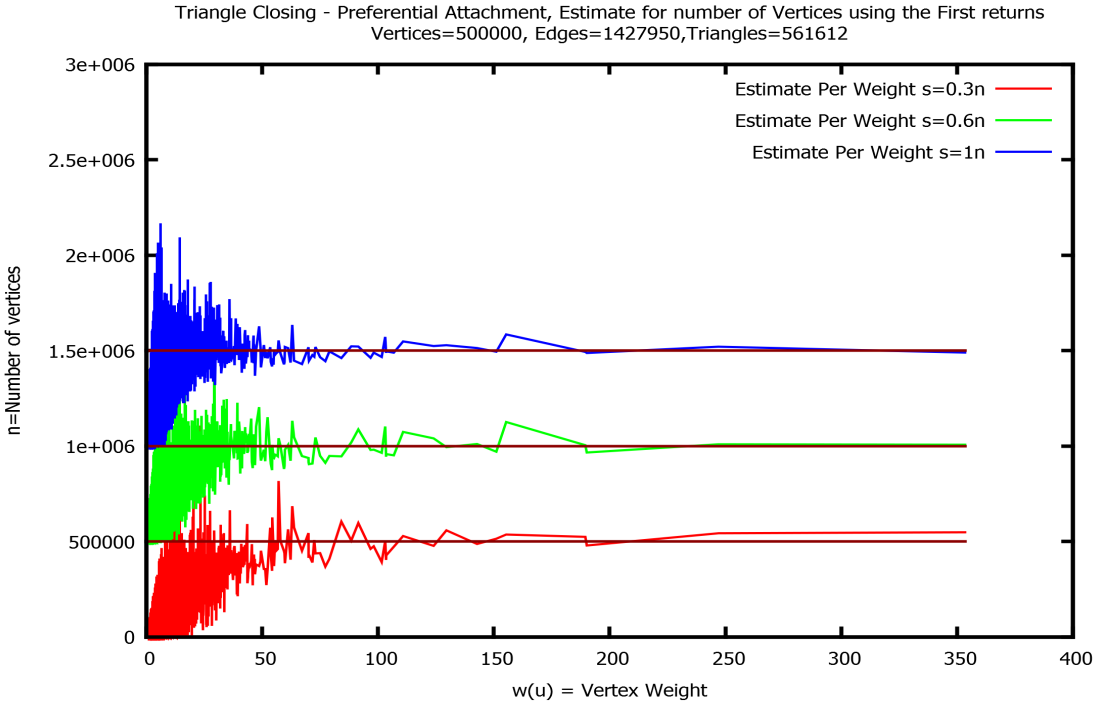
6.1.2 Hybrid model

We use a hybrid triangle closing model which generates graphs as follows. At each step we add a new vertex v with r edges to the existing graph. To add a vertex v, we first attach to an existing vertex x chosen preferentially. The remaining edges $2, \ldots, r$ from v are added as follows. With probability p we attach a vertex chosen by preferential attachment, and with probability $1 - p$ we add an edge from v to a random neighbour of x. Using this approach we are able to control the number of triangles generated while maintaining the power-law degree distribution.

We generated a graph using this model with $n = 0.5 * 10^6, m = 1.4 * 10^6$ and $p = 0.3$. At each step $r = 3$ edges were added. The total number of triangles was $t = 5.61 * 10^6$.

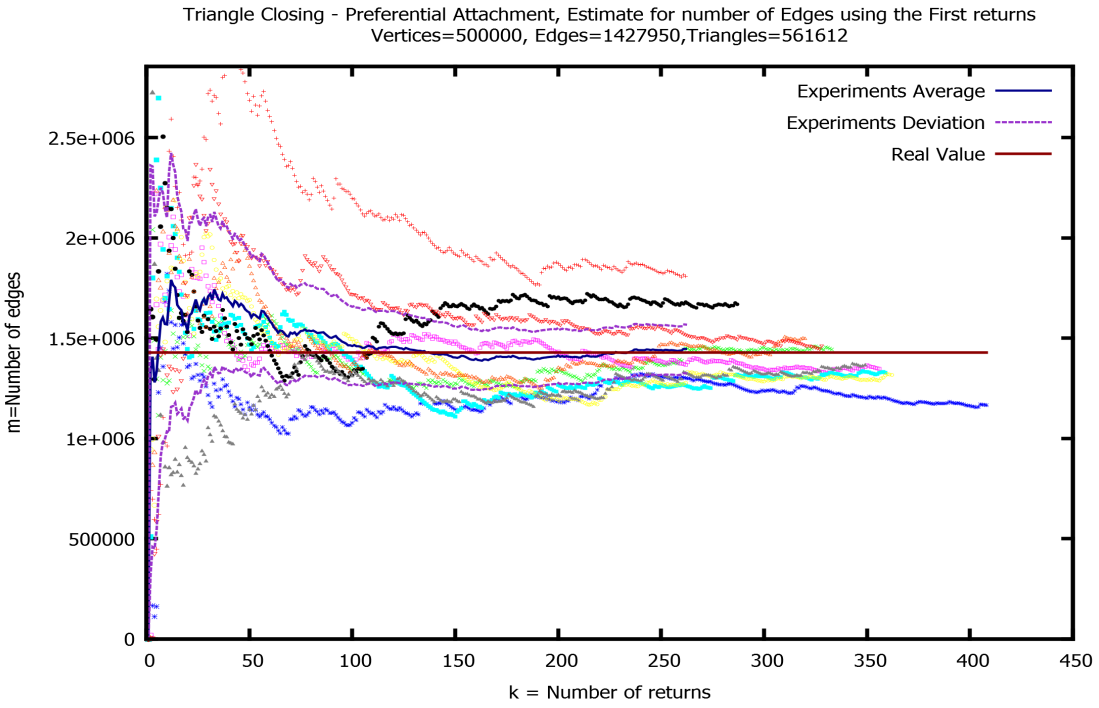


(a) Estimate based on returns to a high weight vertex

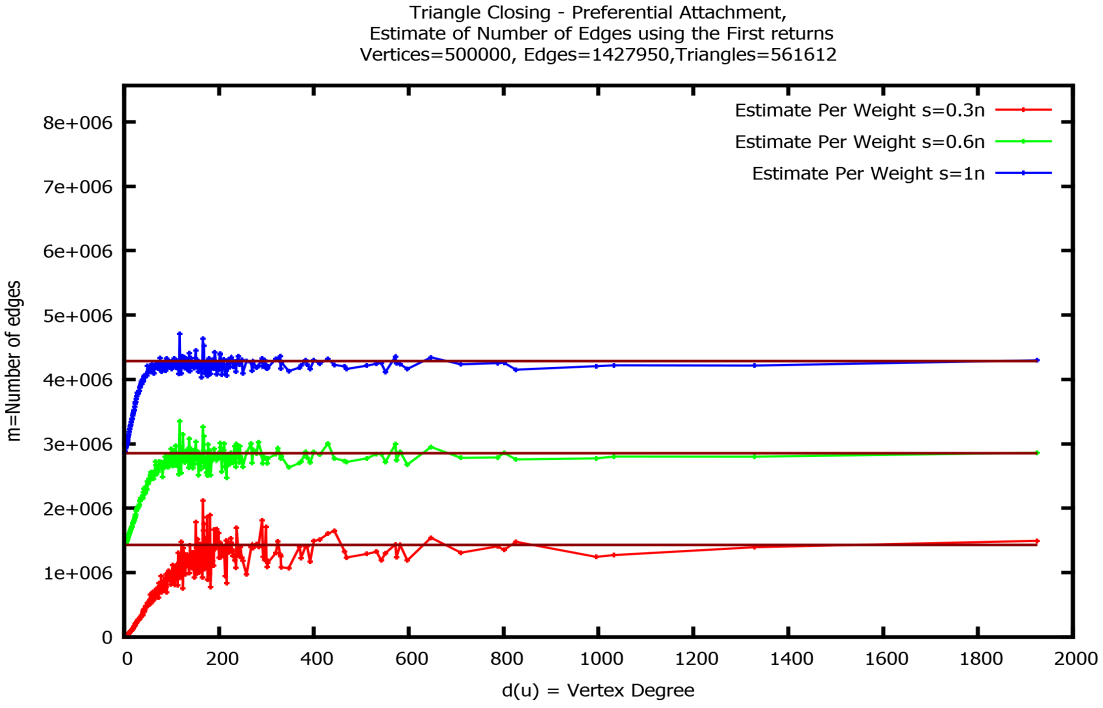


(b) Estimate as a function of vertex weight

Figure 6.5: Vertex estimates for the hybrid triangle closing model based on the first return times of a VRW

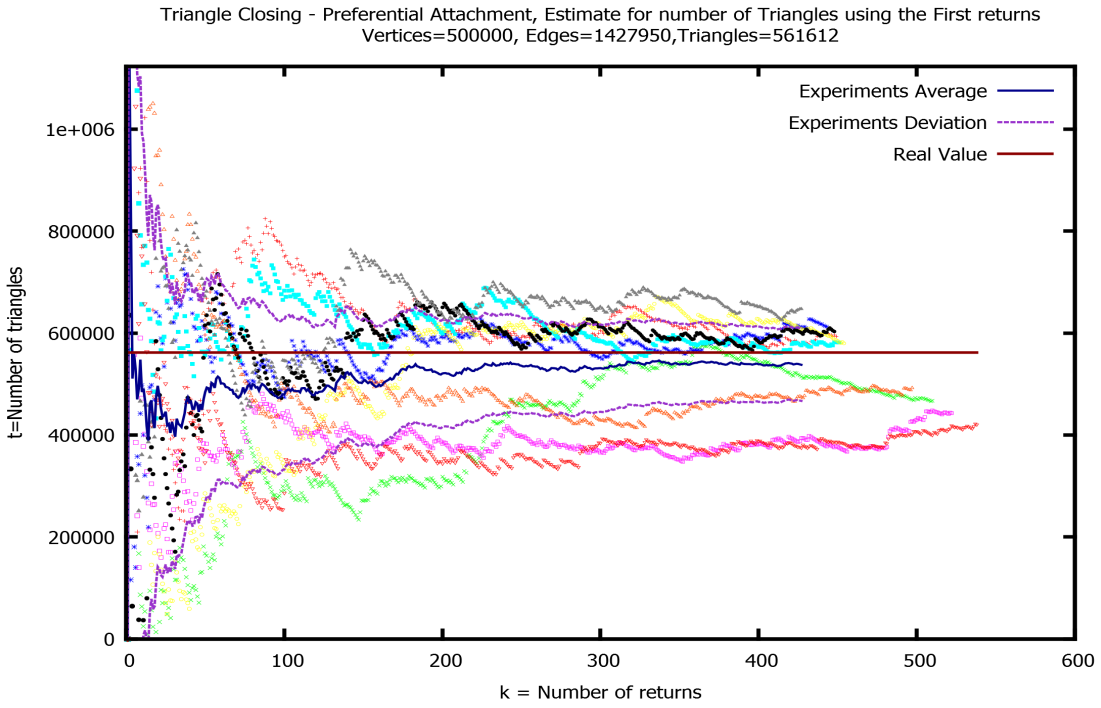


(a) Estimate based on returns to a high weight vertex

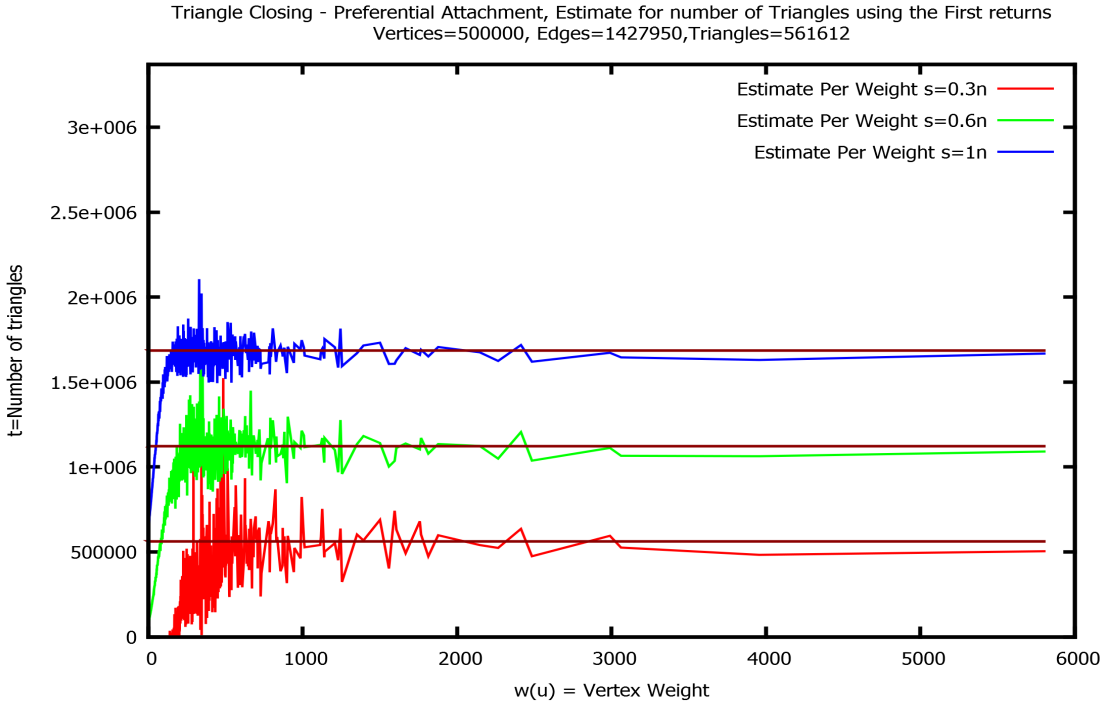


(b) Estimate as a function of vertex weight

Figure 6.6: Edge estimates for the hybrid triangle closing model based on the first return times of a SRW

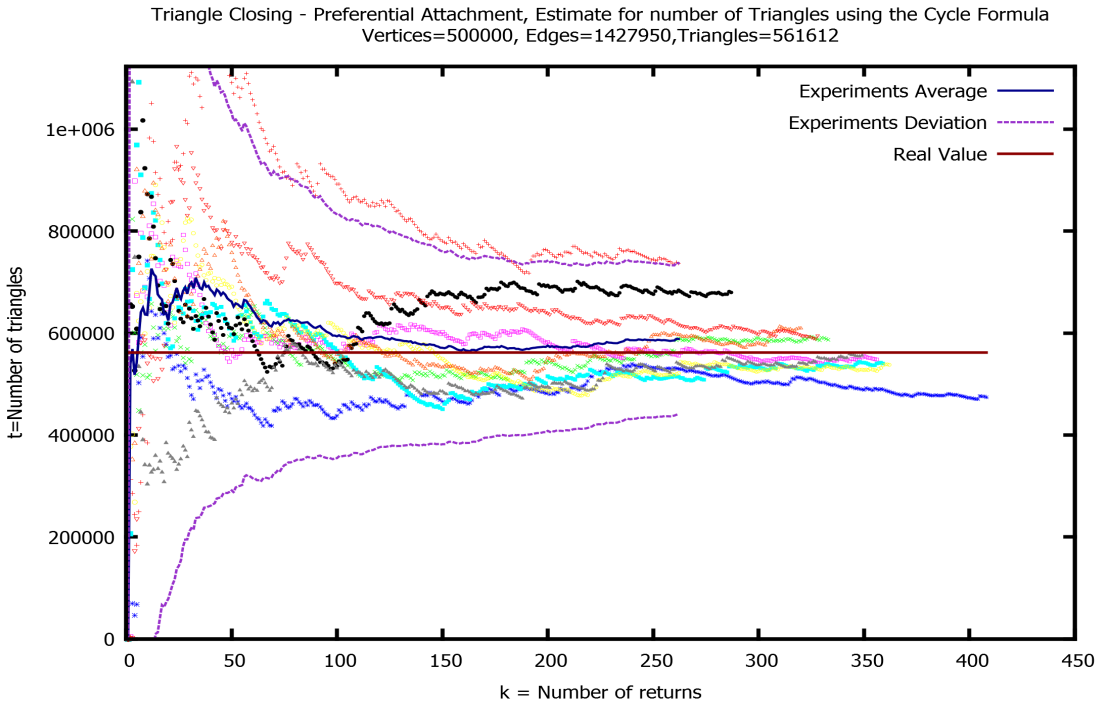


(a) Estimate based on returns to a high weight vertex

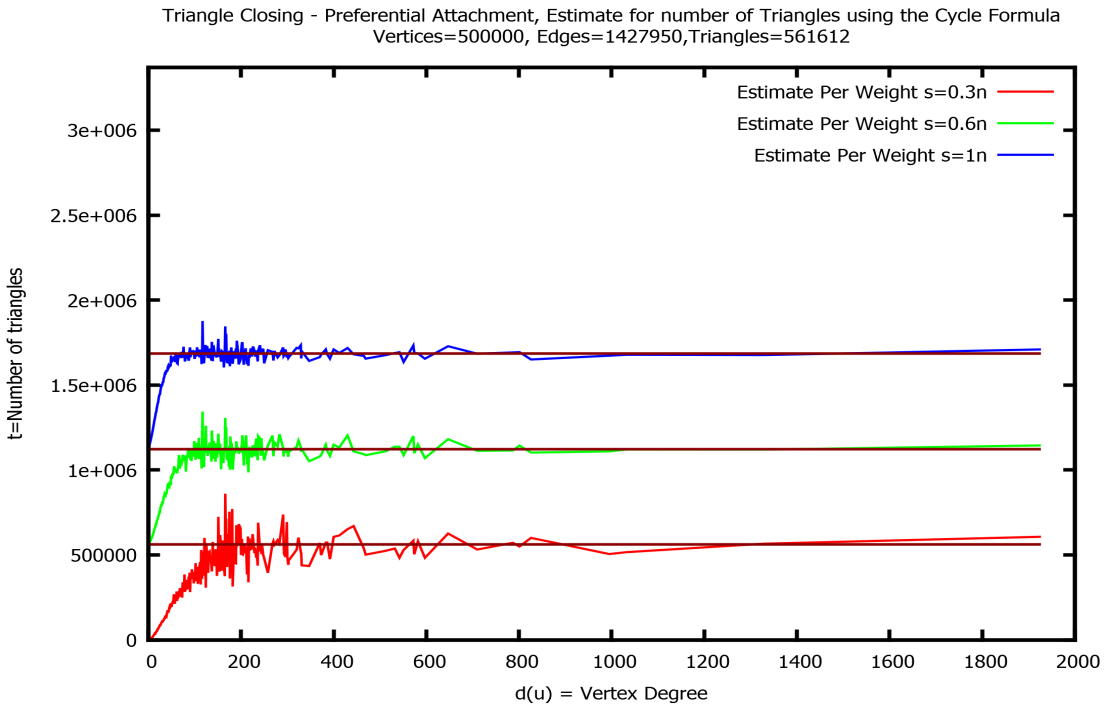


(b) Estimate as a function of vertex weight

Figure 6.7: Triangle estimates for the hybrid triangle closing model based on the first return times of a TRW

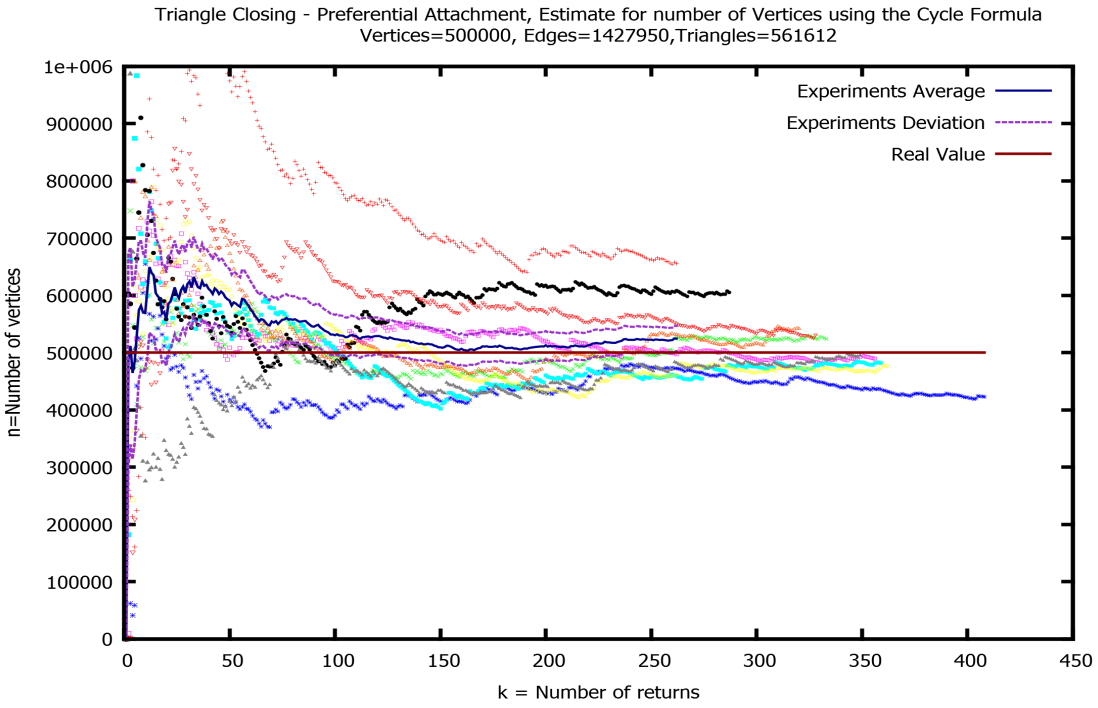


(a) Estimate based on returns to a high weight vertex

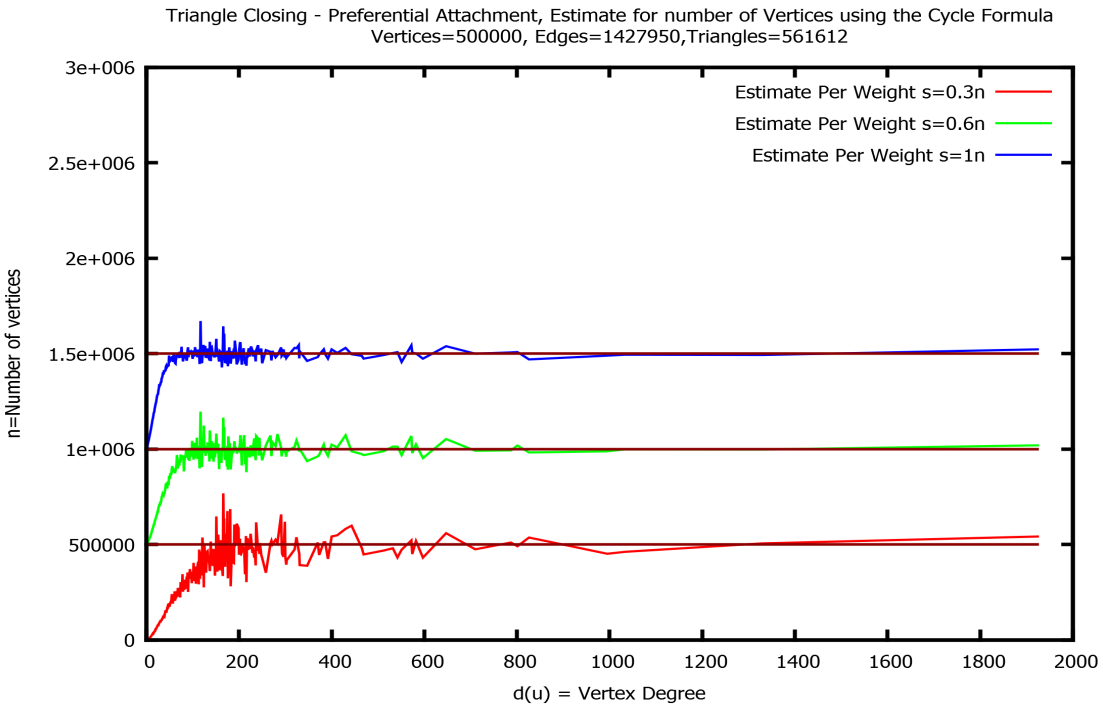


(b) Estimate as a function of vertex weight

Figure 6.8: Triangle estimates for the hybrid triangle closing model based on the CFRP



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.9: Vertex estimates for the hybrid triangle closing model based on the CFRP

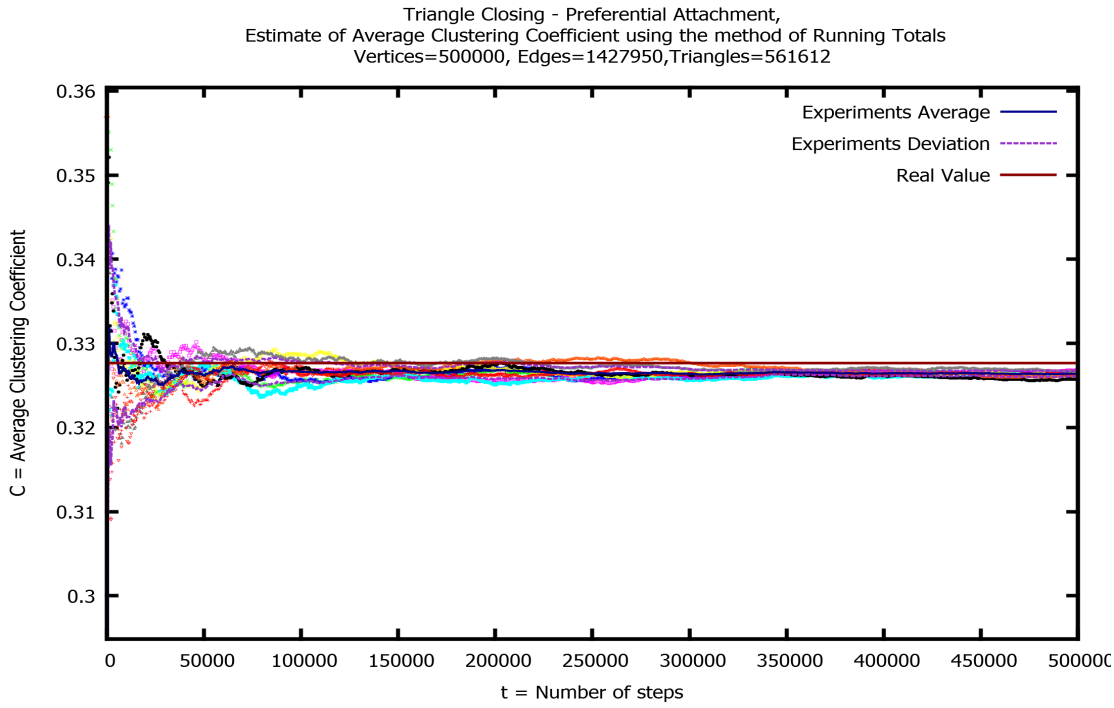
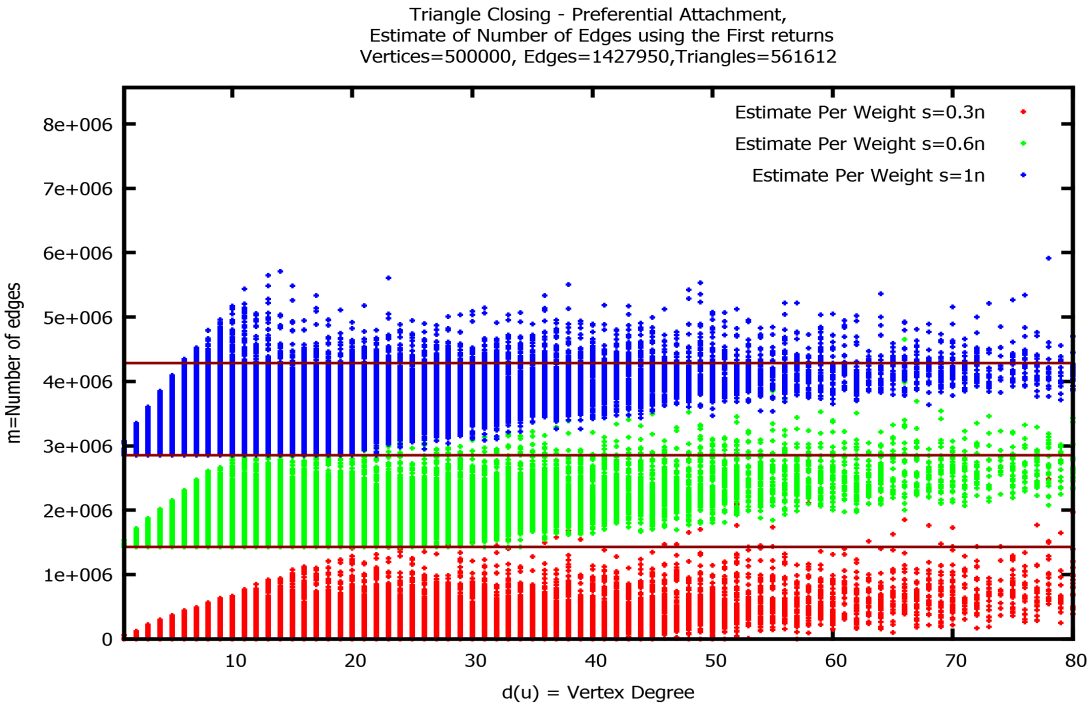
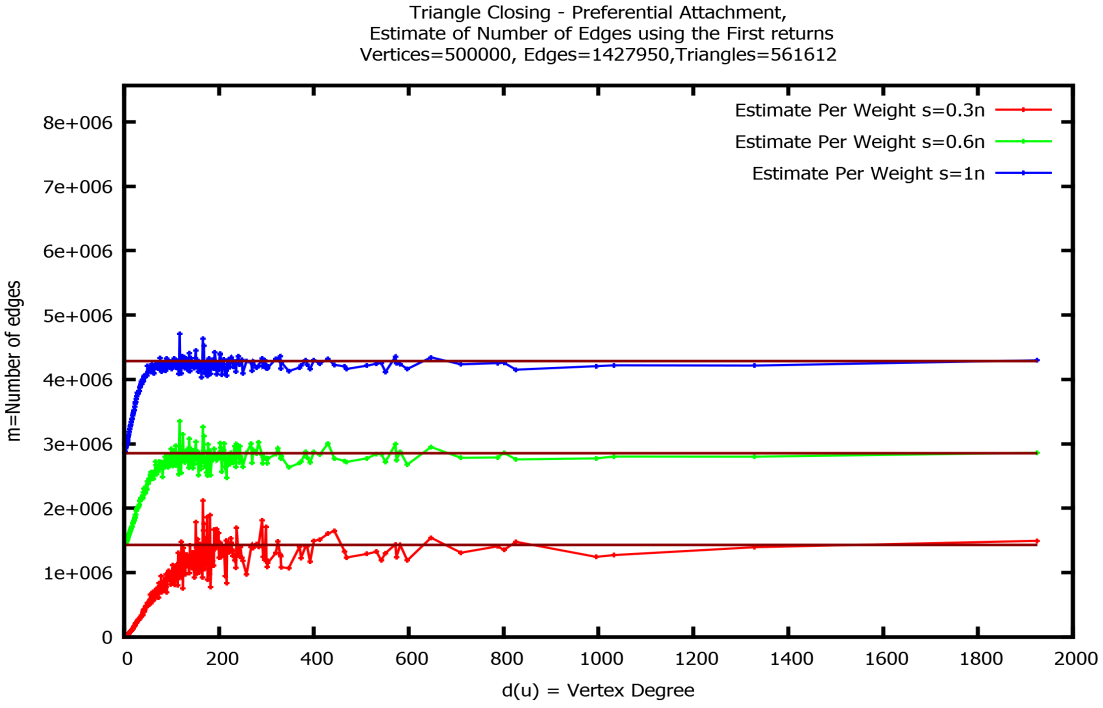


Figure 6.10: Clustering coefficient estimate using the running totals method for estimating network averages (see Section 5.2.5)



(a) Estimates over all vertices (with $d(u) < 80$) sorted by degree



(b) Estimate per vertex weight averaged over all vertices of the same weight.

Figure 6.11: Edge estimates for the hybrid triangle closing model based on the first return times of a SRW

Estimates based on return times. In Figure 6.5 we can see the vertex estimate $\widehat{n}$ obtained using the walk described in Equation (5.21). The edge estimator for this network is given in Figure 6.6. In Figure 6.7 we can see the triangle estimate $\widehat{t}$ obtained using the walk described in Equation (5.22).

In Section 6.1 we mention that the estimate per vertex weight is averaged over all vertices of the same weight. In Figure 6.11 we show what the averaging results in. The left hand side plot shows the same data as the right hand side plot, but not averaged. The weight is limited to $d(u) < 80$. The actual estimates are not concentrated around the average in the lower degrees but seem to converge for higher degree vertices. This further emphasises the importance of starting at a high degree vertex in order to obtain accurate results.

Estimates based on the CFRP In Figure 6.9 we can see the vertex estimate $\widehat{n}$ obtained using the walk described in Equation (5.21). In Figure 6.8 we can see the triangle estimate $\widehat{t}$ obtained using the walk described in Equation (5.23).

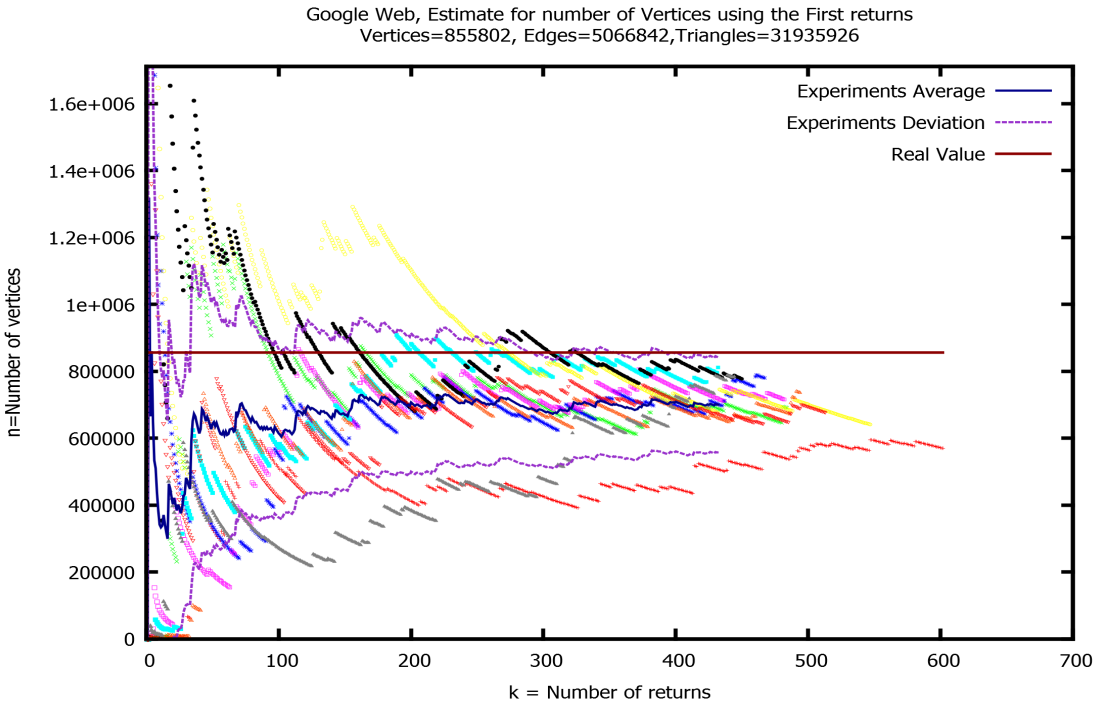
In all cases the estimators appear stable and convergent, and the comments made for the preferential attachment model apply equally in this case.

6.1.3 Google Web Sample

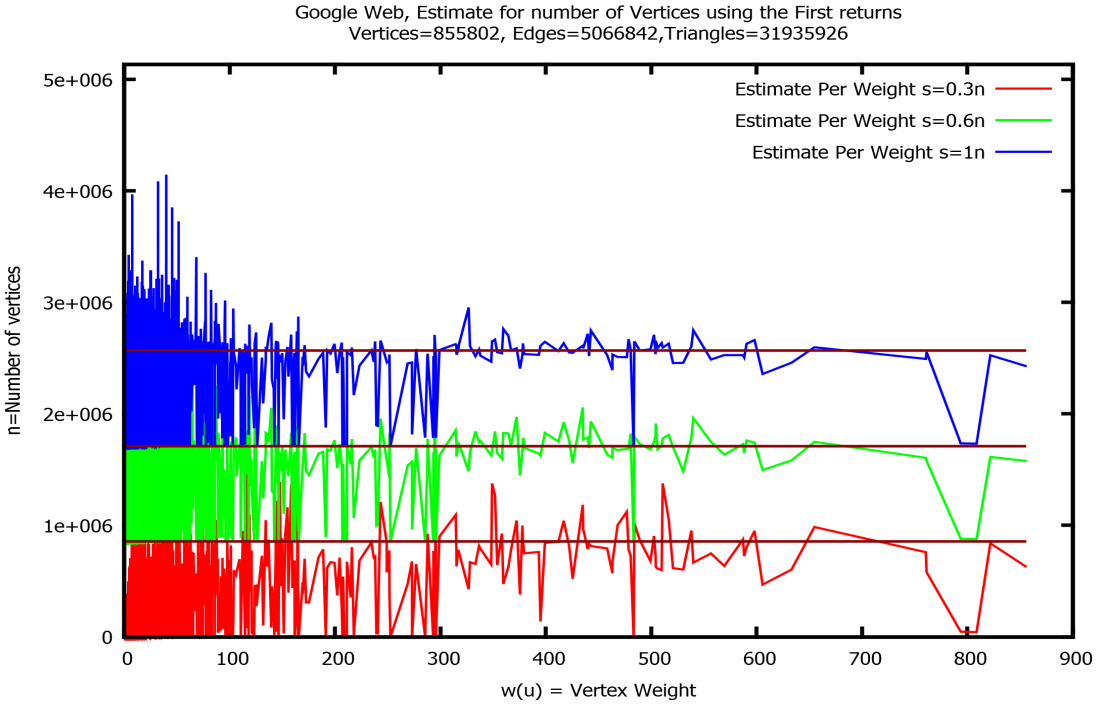
A sample from the Google web graph released for the purposes of the Google programming contest in 2002. This dataset can be found in [83] and consists of 855802 vertices and 5066842 edges and 31935926 triangles if multi-triangles are counted or 13356298 simple triangles. Since this graph is not connected, these quantities refer to its largest WCC.

For this network the estimates converged more slowly than in the generated test graphs, with much more variation around the expected values as the walks progressed.

Estimate of number of vertices. In Figure 6.12 the right hand figure plots the estimate of the number of vertices over time for the highest weight vertex accessed by the walk. Although reasonably convergent, the rate of convergence is much slower than

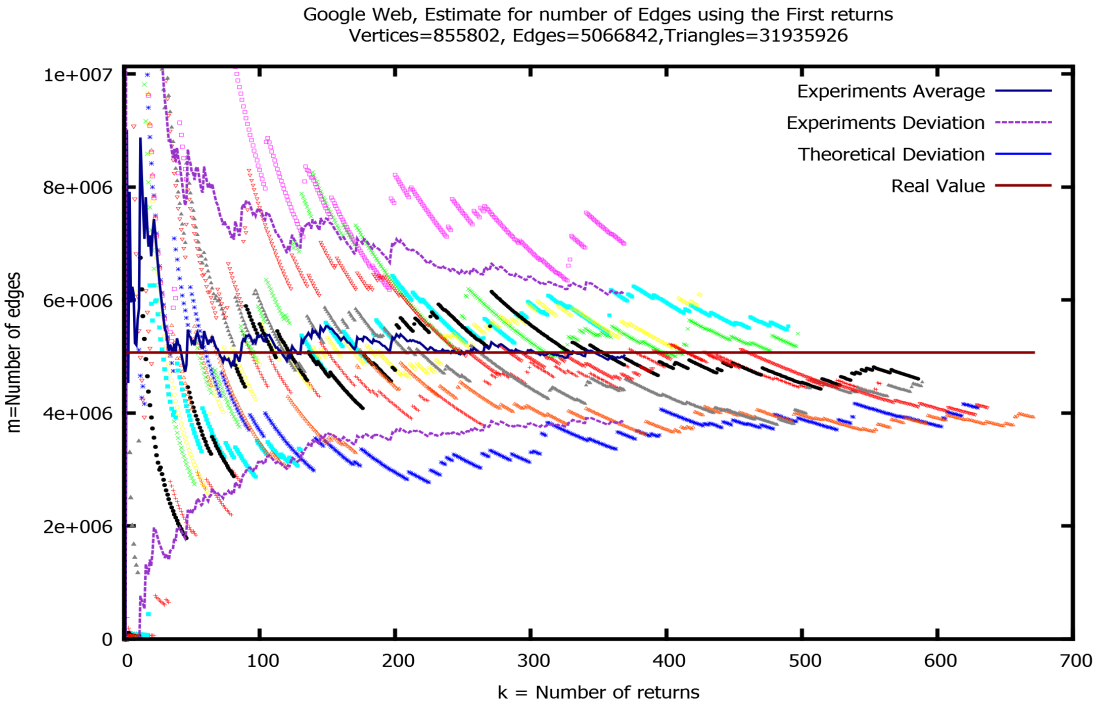


(a) Estimate based on returns to a high weight vertex

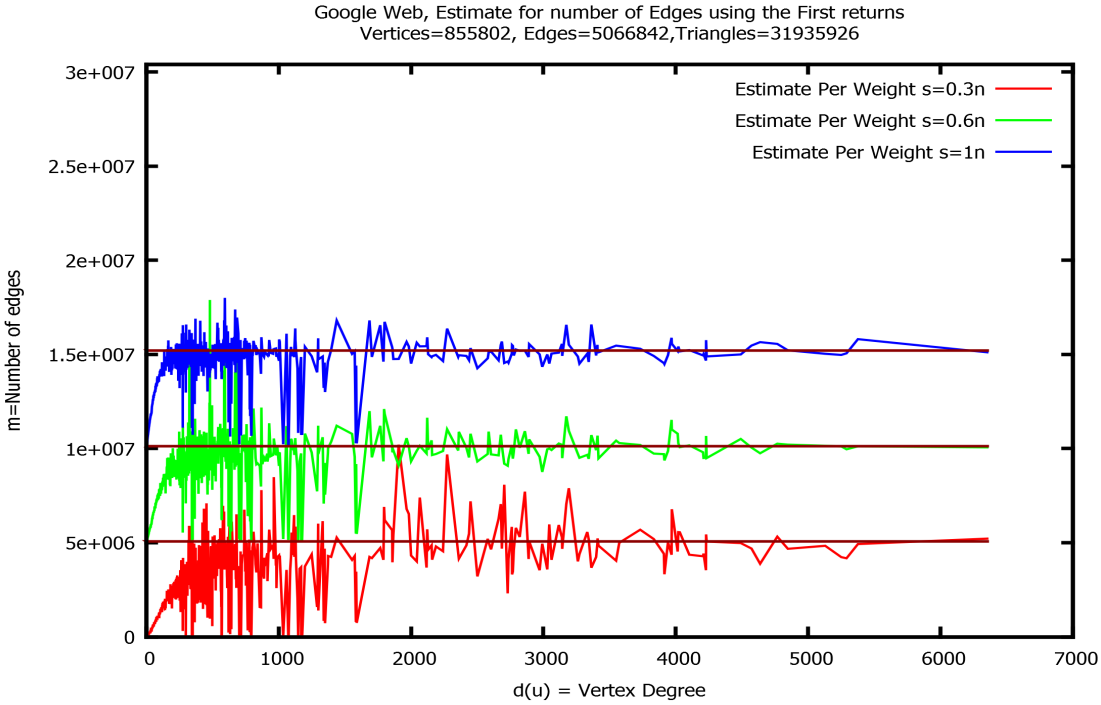


(b) Estimate as a function of vertex weight

Figure 6.12: Vertex estimates for the Google Web graph sample based on the first return times of a VRW

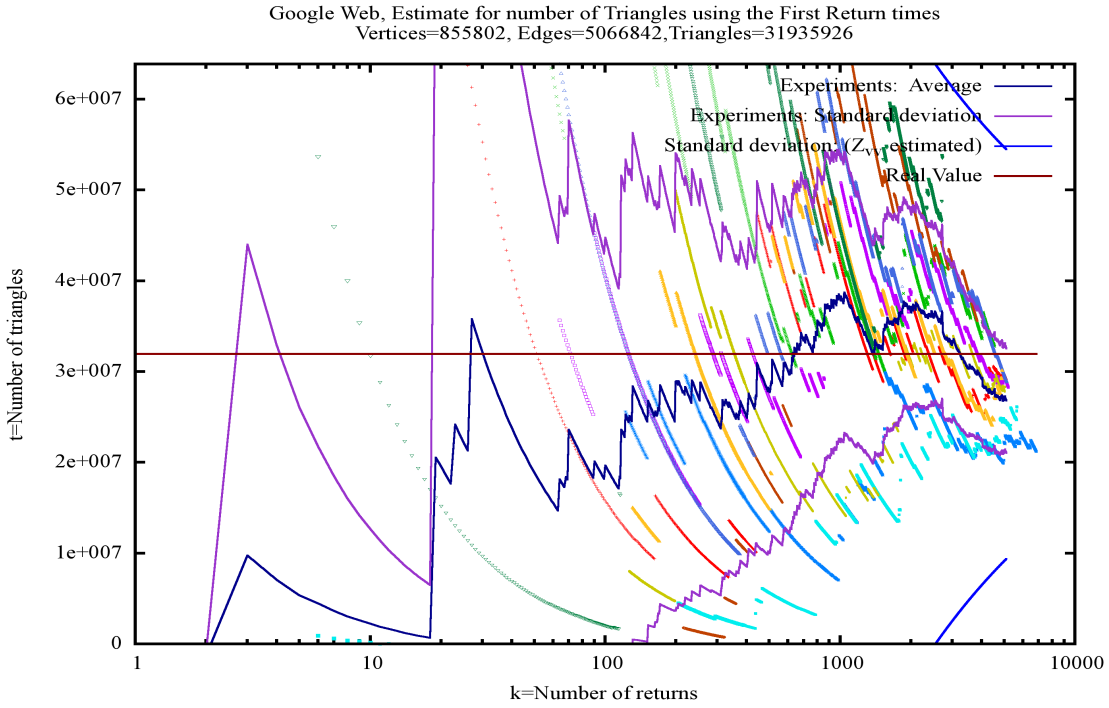


(a) Estimate based on returns to a high weight vertex

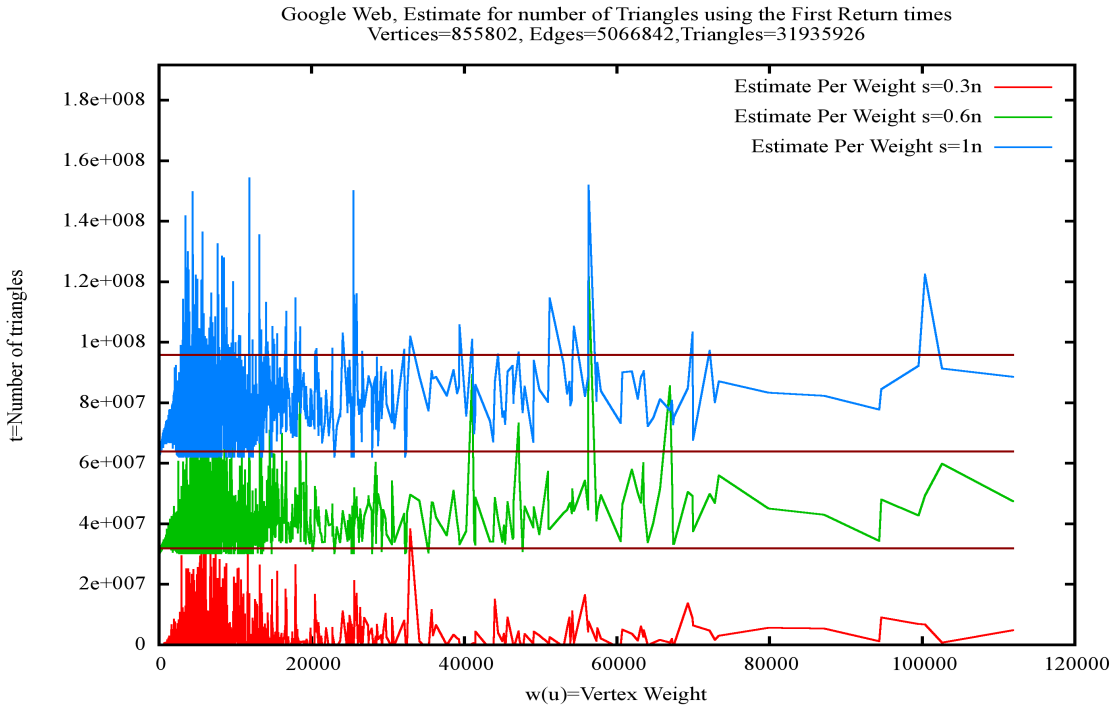


(b) Estimate as a function of vertex weight

Figure 6.13: Edge estimates for the Google Web graph sample based on the first return times of a SRW

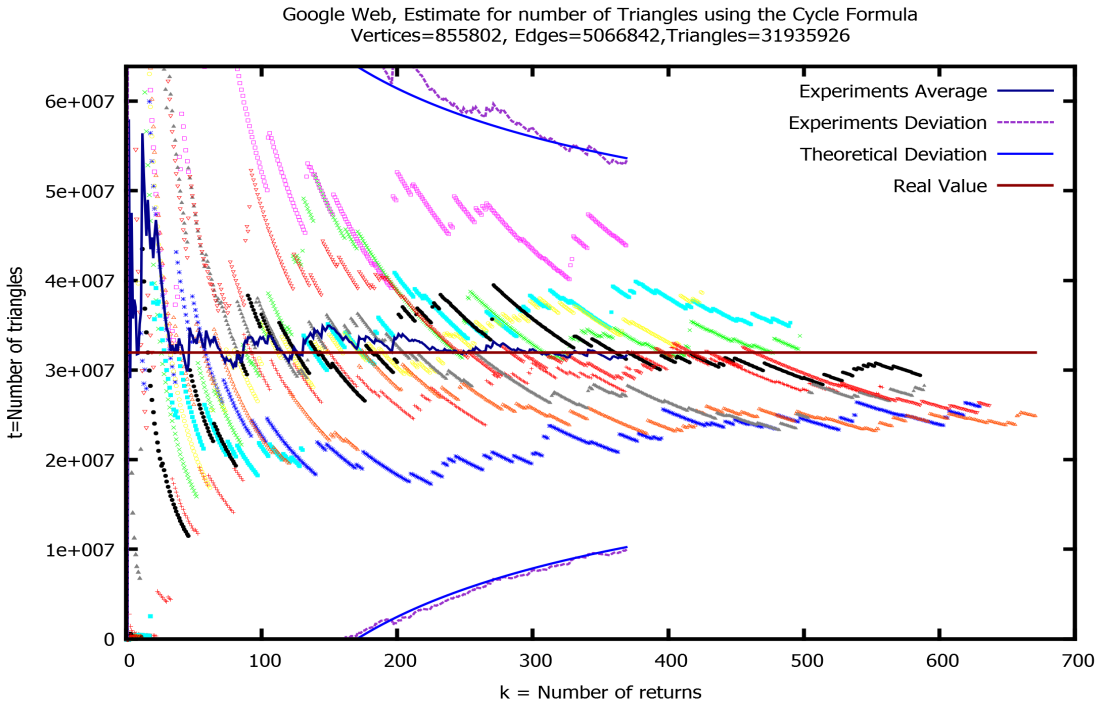


(a) Estimate based on returns to a high weight vertex

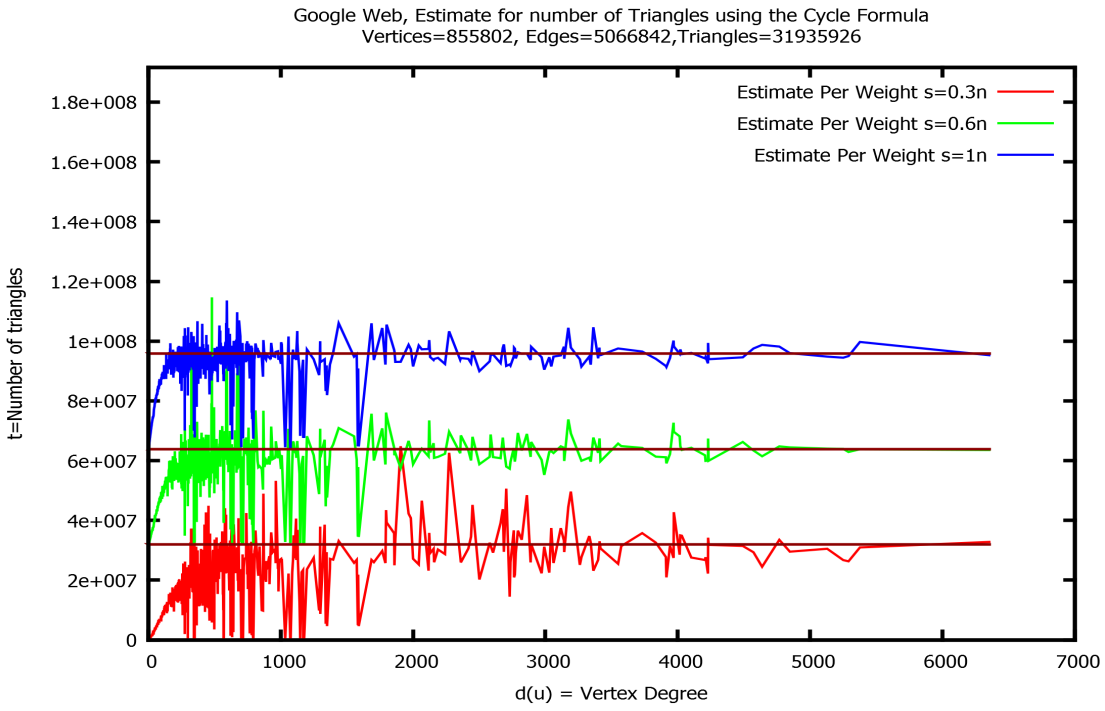


(b) Estimate as a function of vertex weight

Figure 6.14: Triangle estimates for the Google Web graph sample based on the first return times of a TRW

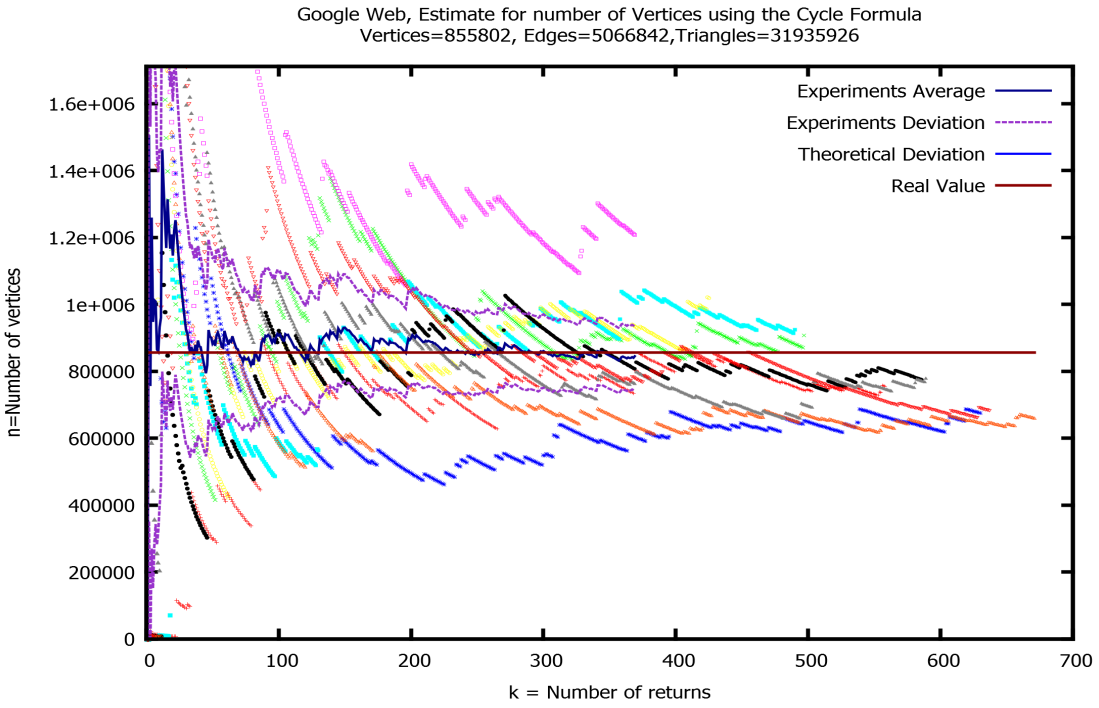


(a) Estimate based on returns to a high weight vertex

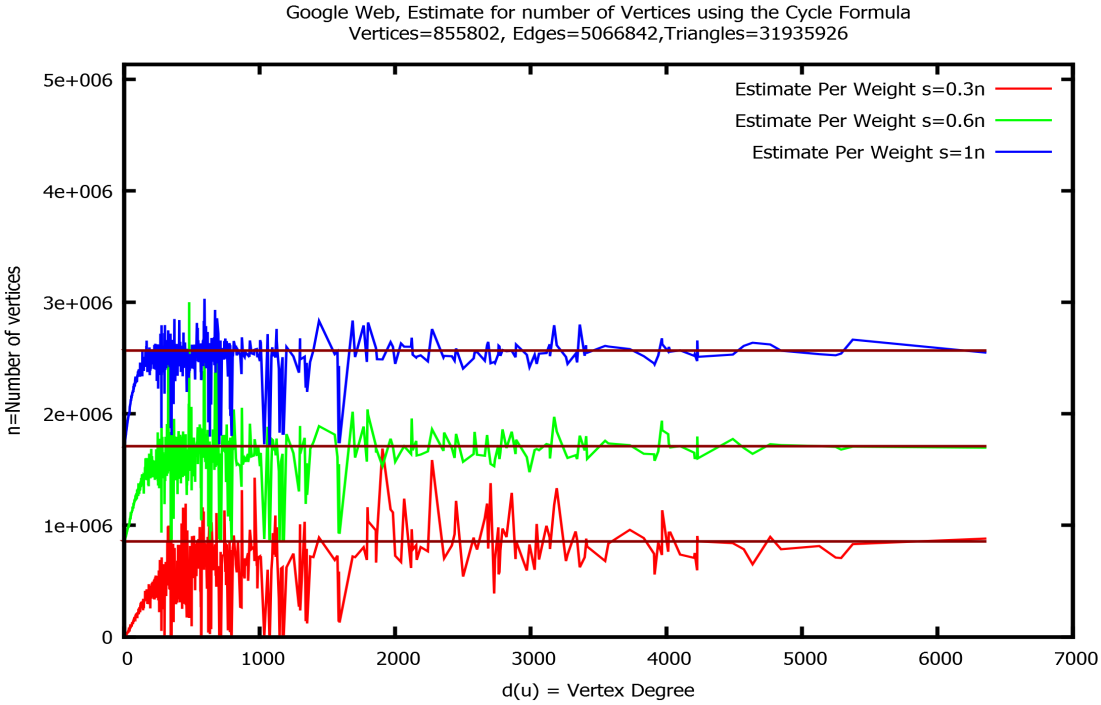


(b) Estimate as a function of vertex weight

Figure 6.15: Triangle estimates for the Google Web graph sample based on the CFRP



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.16: Vertex estimates for the Google Web graph sample based on the CFRP

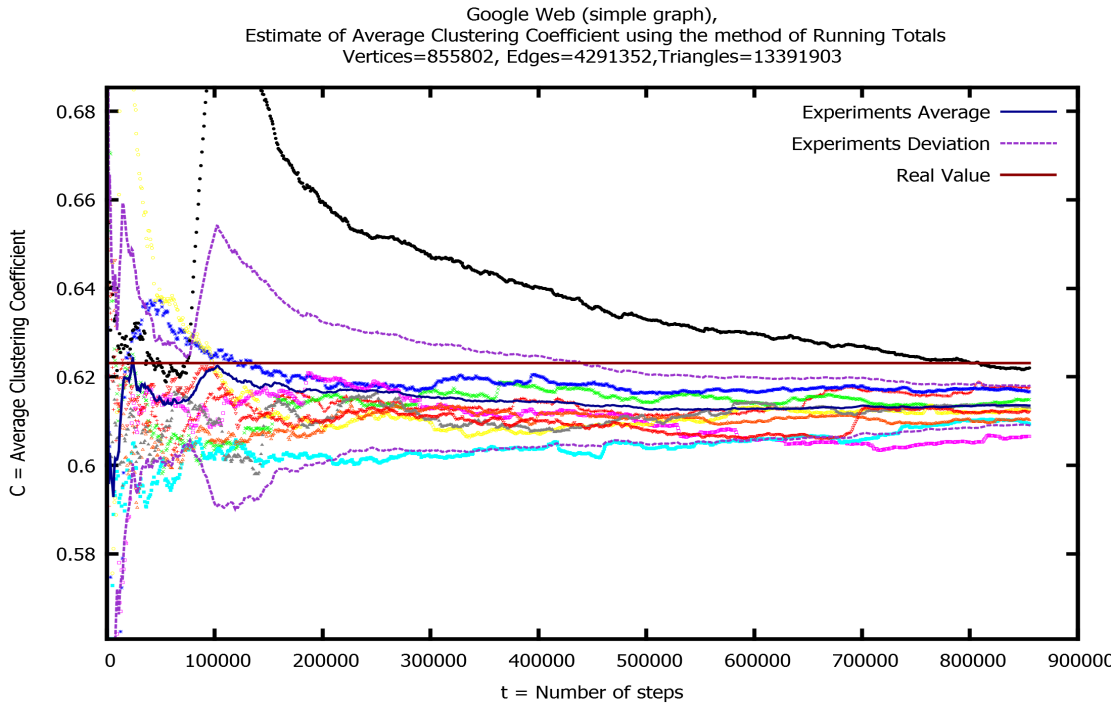


Figure 6.17: Google Web: Clustering coefficient estimate using the running totals method for estimating network averages

in the randomly generated test graphs. The left hand plot shows the convergence of the estimator based on returns to the highest weight vertex. Note the large variability of the results between each different run of the experiments. In Figure 6.16 we see similar vertex estimates $\widehat{n}$ obtained using the CFRP. In this case the underlying walk was a SRW.

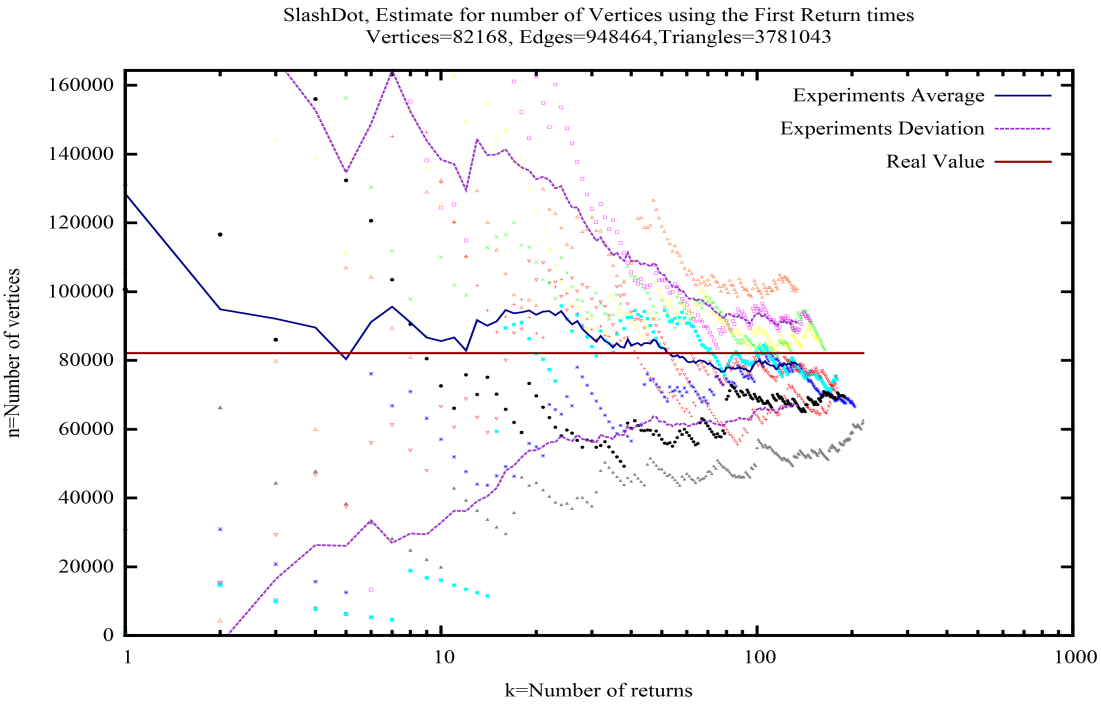
Estimate of number of edges. In Figure 6.13 we see the edge estimate $\widehat{m}$ of this graph using a simple random walk. The performance of this walk appears better than the weighted counterpart used above to estimate the number of vertices.

Estimate of number of triangles. In Figure 6.14a we see the triangle estimate $\widehat{t}$ obtained from the highest weight vertex, using the walk described in Equation (5.22), Although apparently convergent, there is considerable fluctuation. In Figure 6.14b we can see the triangle estimate per vertex weight. The estimate is still below the true network value at $0.3n$ steps and $0.6n$. By n steps the result seems better, but considerable variability in the data still remains. We ran these experiments for $5n$ steps to get reasonably accurate results.

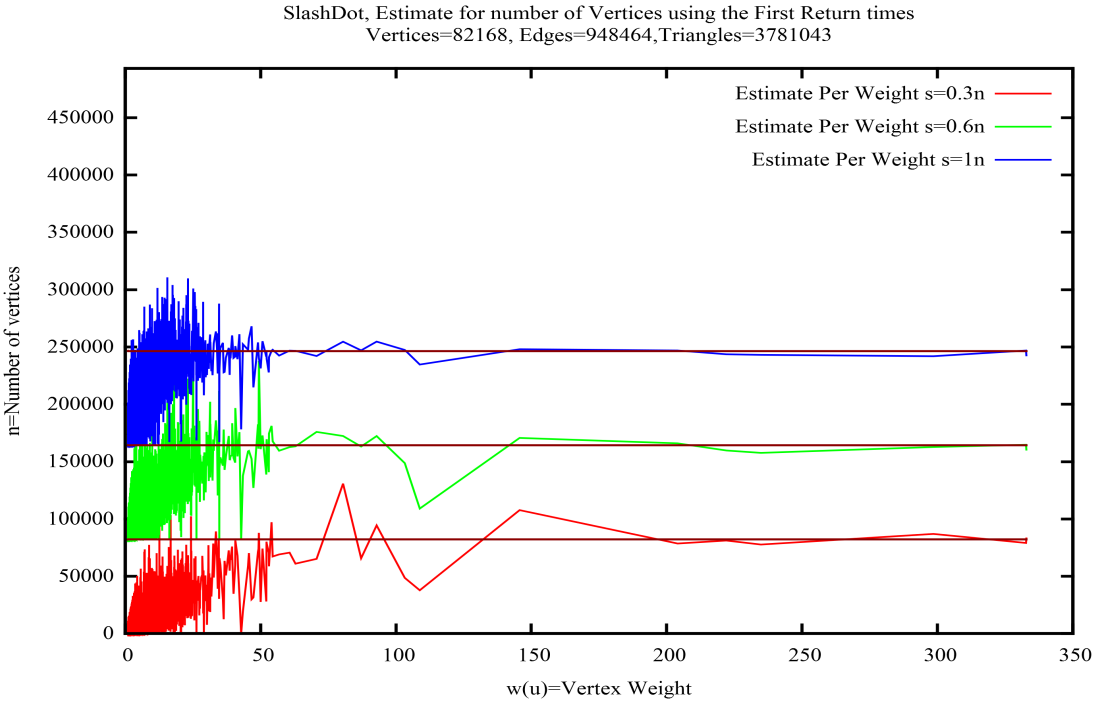
Estimate of the clustering coefficient Based on the method of the running totals and using the procedure described in Section 5.2.5 we have estimated the average clustering coefficient in this network. The results can be seen in Figure 6.17. We note that the underlying graph in this case is the same as above, but with all multi-edges and loops removed. This estimator seems to converge rapidly.

6.1.4 SlashDot Zoo Network

SlashDot is a website which posts technology related news and events. It is owned by Dice Holdings, Inc. and ran by a handful of editors and coders [97]. In 2002 SlashDot introduced a social network type of feature called the *SlashDot Zoo* where users could tag other users as friends or foes. We run our experiments on this graph, ignoring edge directions and whether or not the edges are friend or foe links. This dataset can be found in [83] and consists of 82168 vertices, 870161 edges and 602592 simple triangles and is

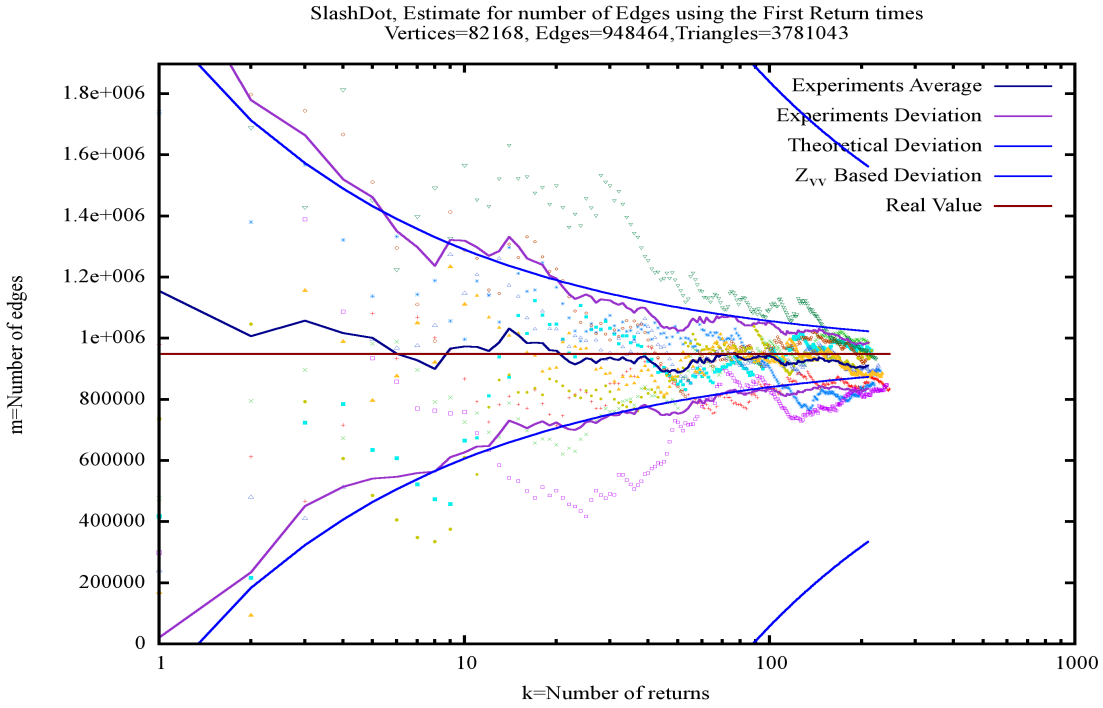


(a) Estimate based on returns to a high weight vertex

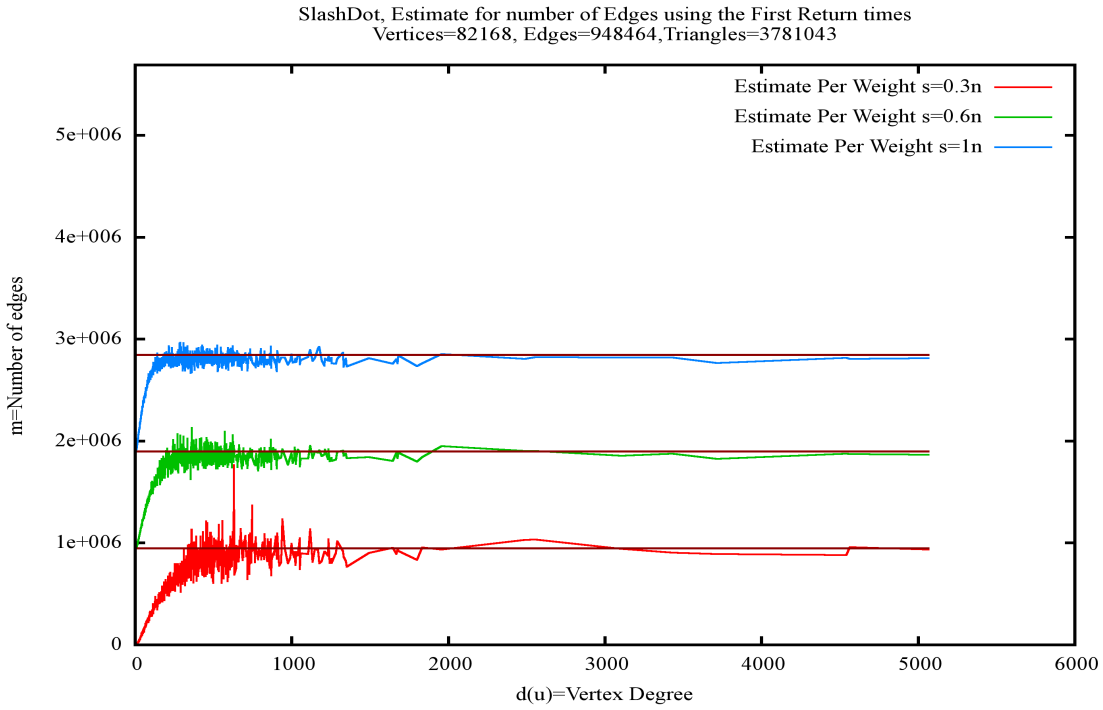


(b) Estimate as a function of vertex weight

Figure 6.18: Vertex estimates for the SlashDot Zoo OLSN based on the first return times of a VRW

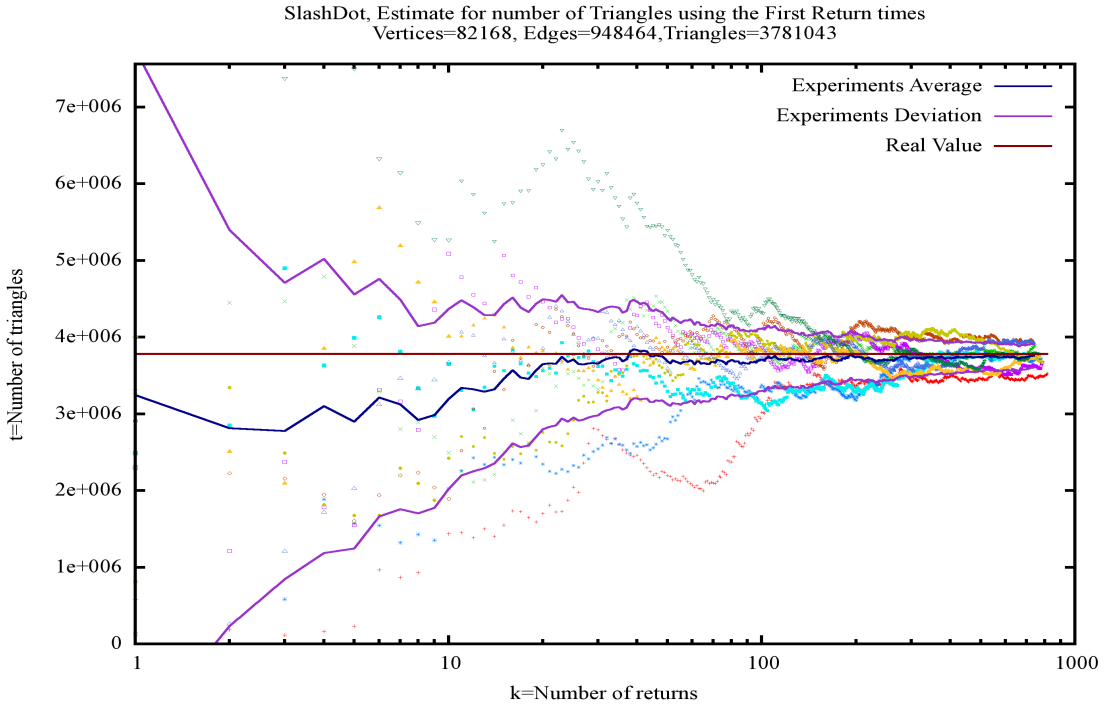


(a) Estimate based on returns to a high weight vertex

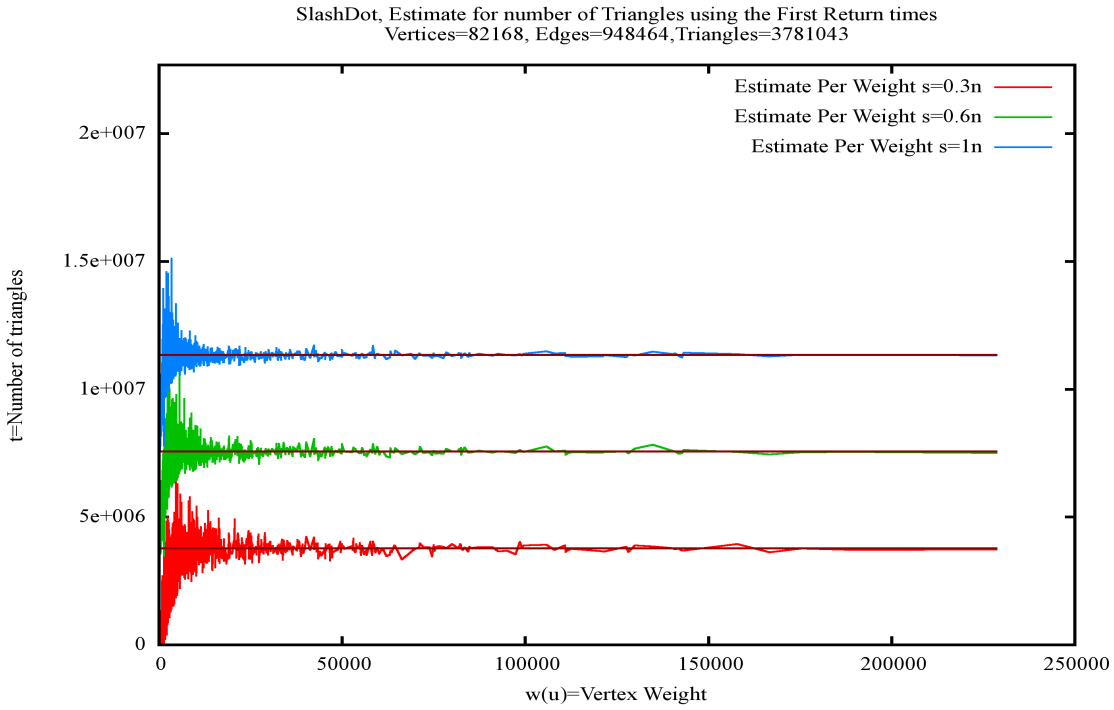


(b) Estimate as a function of vertex weight

Figure 6.19: Edge estimates for the SlashDot Zoo OLSN based on the first return times of a SRW

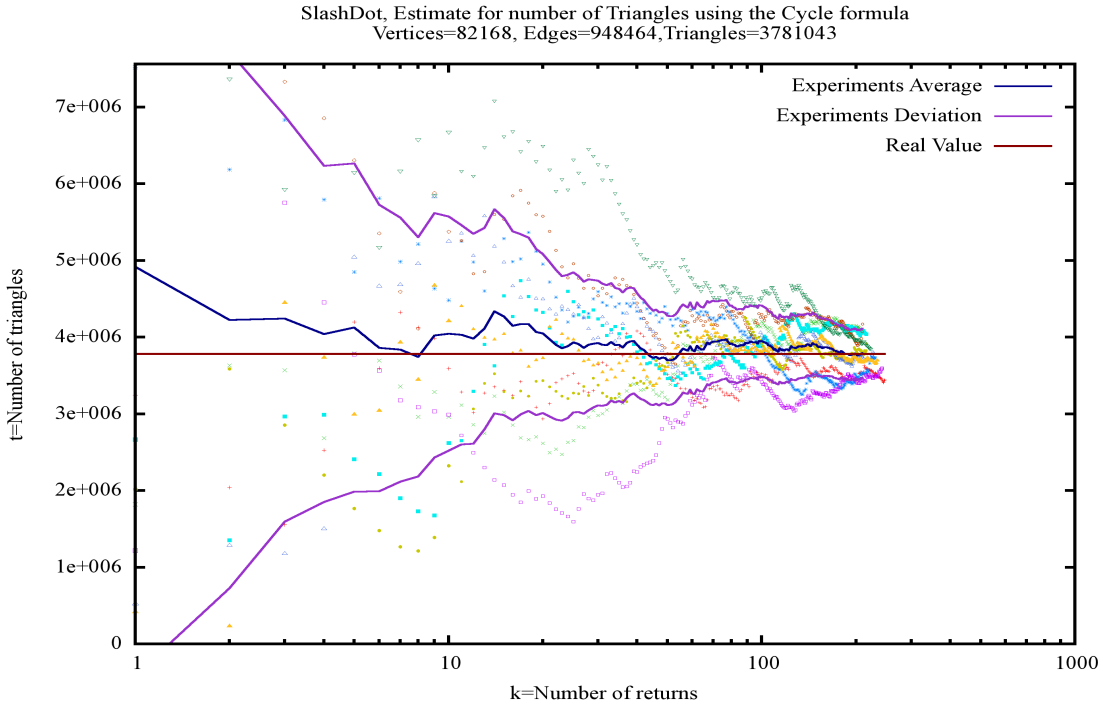


(a) Estimate based on returns to a high weight vertex

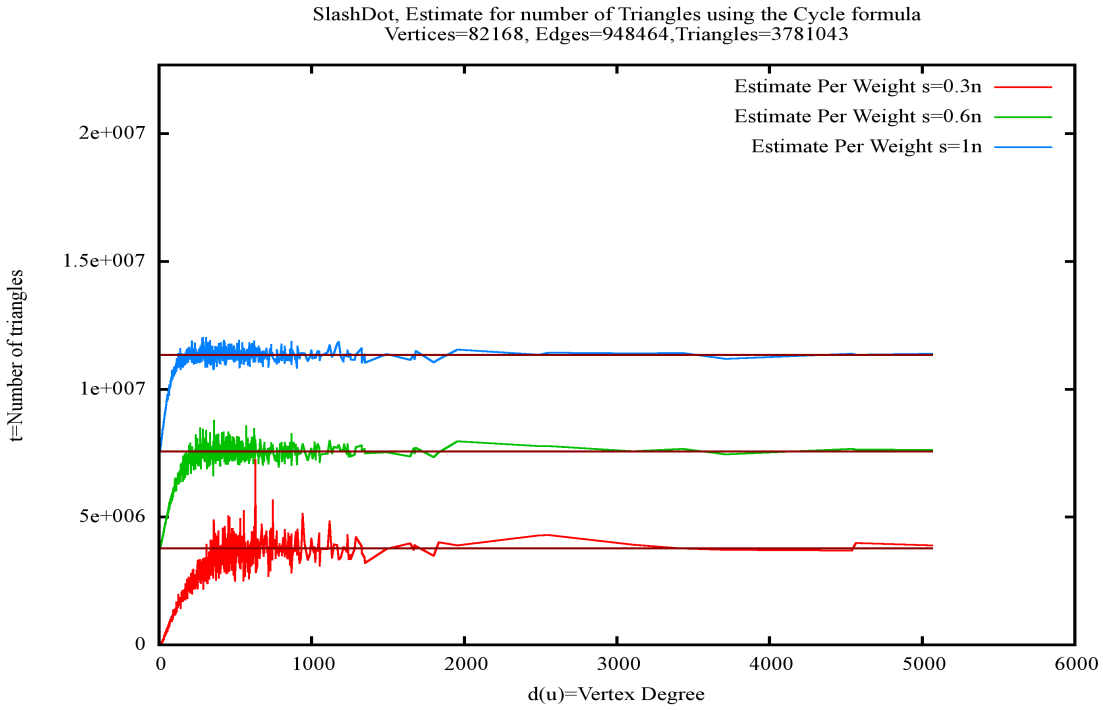


(b) Estimate as a function of vertex weight

Figure 6.20: Triangle estimates for the SlashDot Zoo OLSN based on the first return times of a TRW

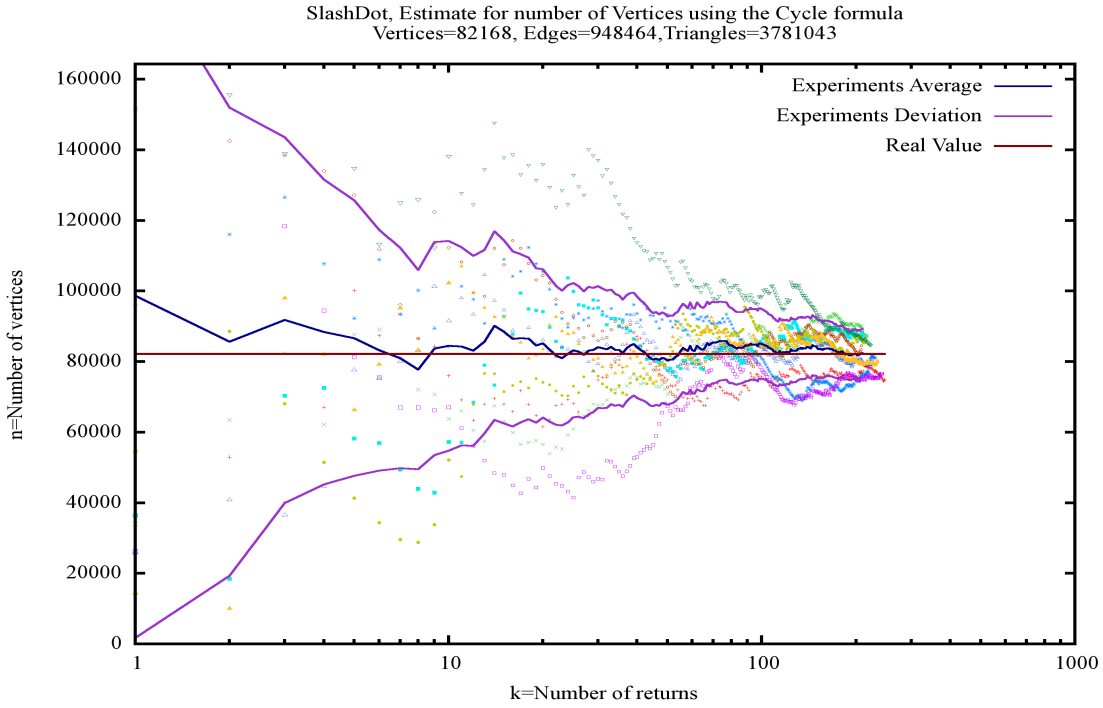


(a) Estimate based on returns to a high weight vertex

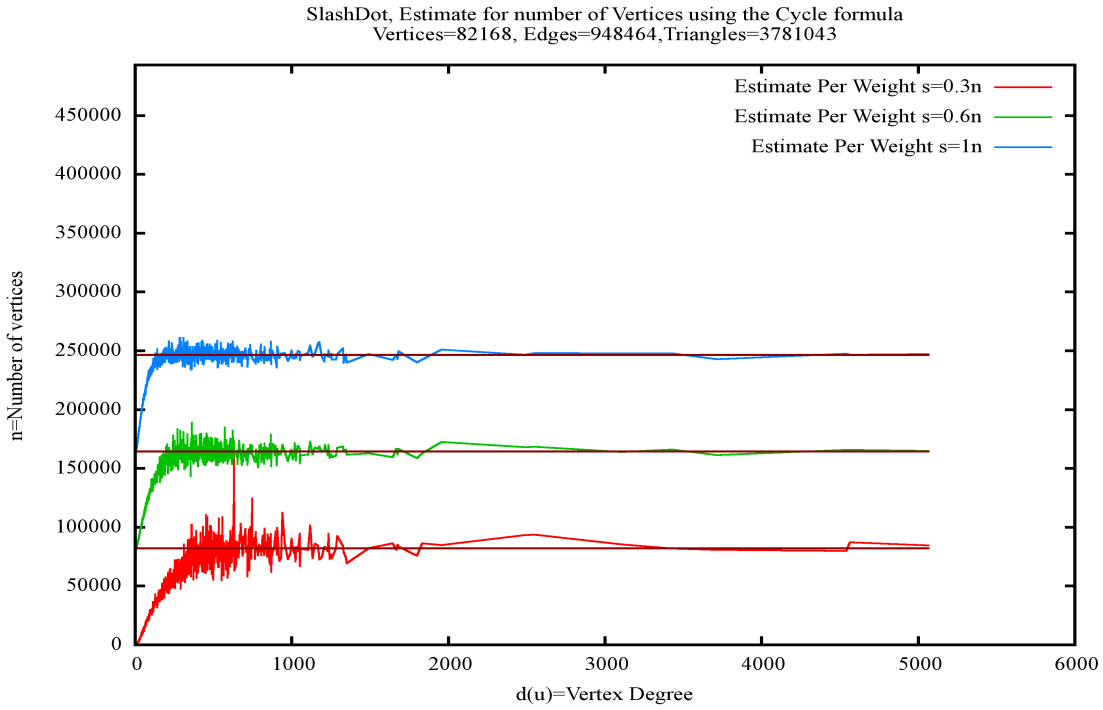


(b) Estimate as a function of vertex weight

Figure 6.21: Triangle estimates for the SlashDot Zoo OLSN based on the CFRP



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.22: Vertex estimates for the SlashDot Zoo OLSN based on the CFRP

weakly connected. However due to multi-edges we have found the number of non-simple triangles to be 4840912. For this network the estimates converged generally well, slower than in the generated test graphs, but faster than in the case of the Google Web graph sample.

Estimate of number of vertices. In Figure 6.18 the left had figure plots the estimate of the number of vertices over time for the highest weight vertex accessed by the walk. The rate of convergence is slower than in the randomly generated test graphs, however it is significantly faster than the Google web sample. In Figure 6.22 we see the vertex estimate $\widehat{n}$ for all sampled vertices, averaged over the vertices of the same weight.

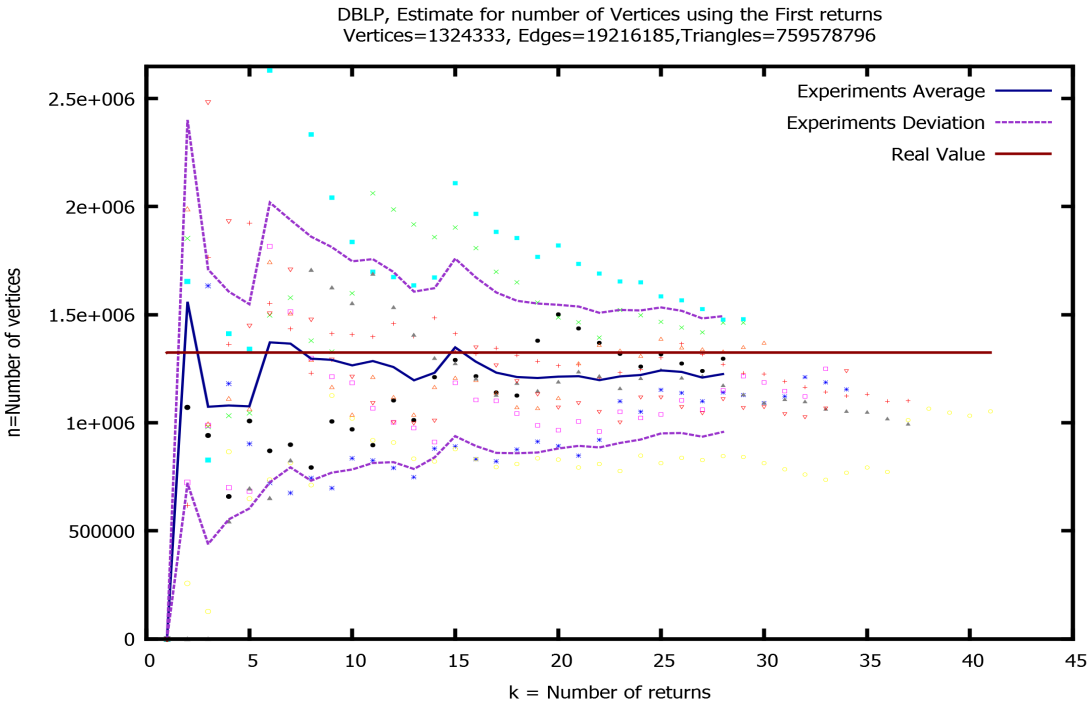
Estimate of number of edges. In Figure 6.19 we see the edge estimate $\widehat{m}$ of this graph using a simple random walk. The performance of this walk appears better than the weighted counterpart used above to estimate the number of vertices. It converges slower than the simulated graphs, but faster than the web graph sample of Google.

Estimate of number of triangles. In Figure 6.20 we see the triangle estimate $\widehat{t}$ obtained from the highest weight vertex, using the walk described in Equation (5.22). In the left hand side of Figure 6.20 we see the estimate based on returns on a single high weight vertex. In the right hand side we see the estimate when we group vertices according to their weight. The estimate in the second case is reasonable even at $0.3n$ steps.

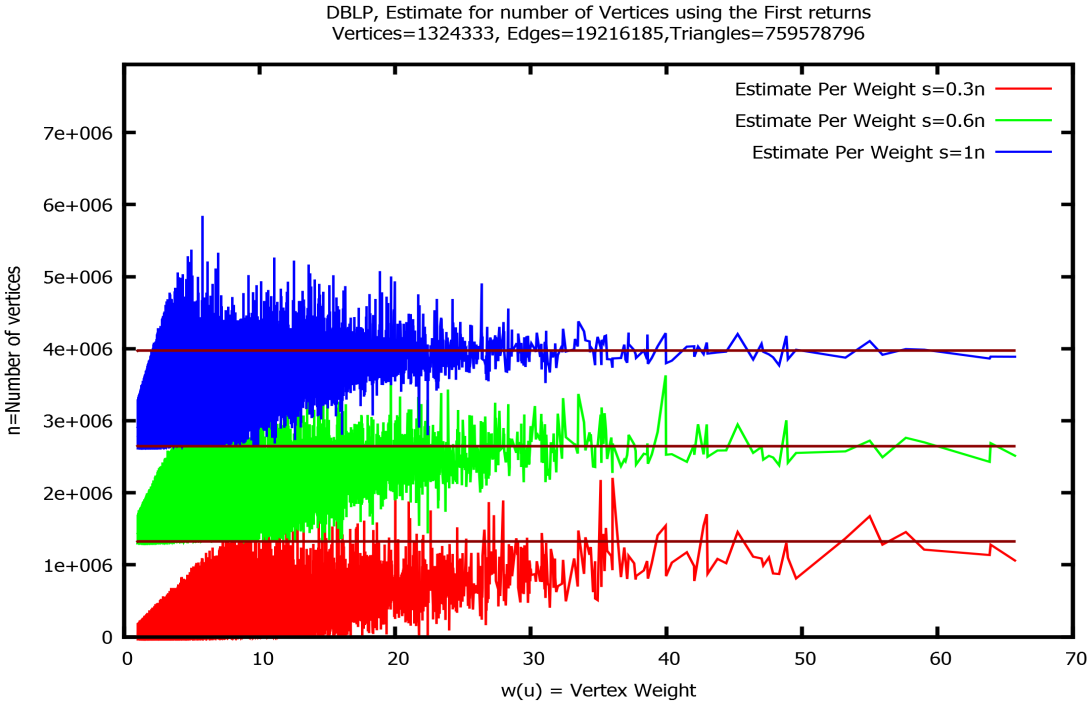
6.1.5 DBLP Co-Author Network

The DBLP website is an online database which contains information about papers published in the broader area of computer science. It is maintained by the DBLP team in the University of Trier. All the data held in the database is openly available under the ODC-BY 1.0 licence[4] and can be found at [124]. For the purposes of our experiments we obtained the data and extracted the Co-Author graph, i.e. if an author had co-authored a paper with another author then an edge was added between them. If an author had

[4] http://opendatacommons.org/licensees/by/1.0/

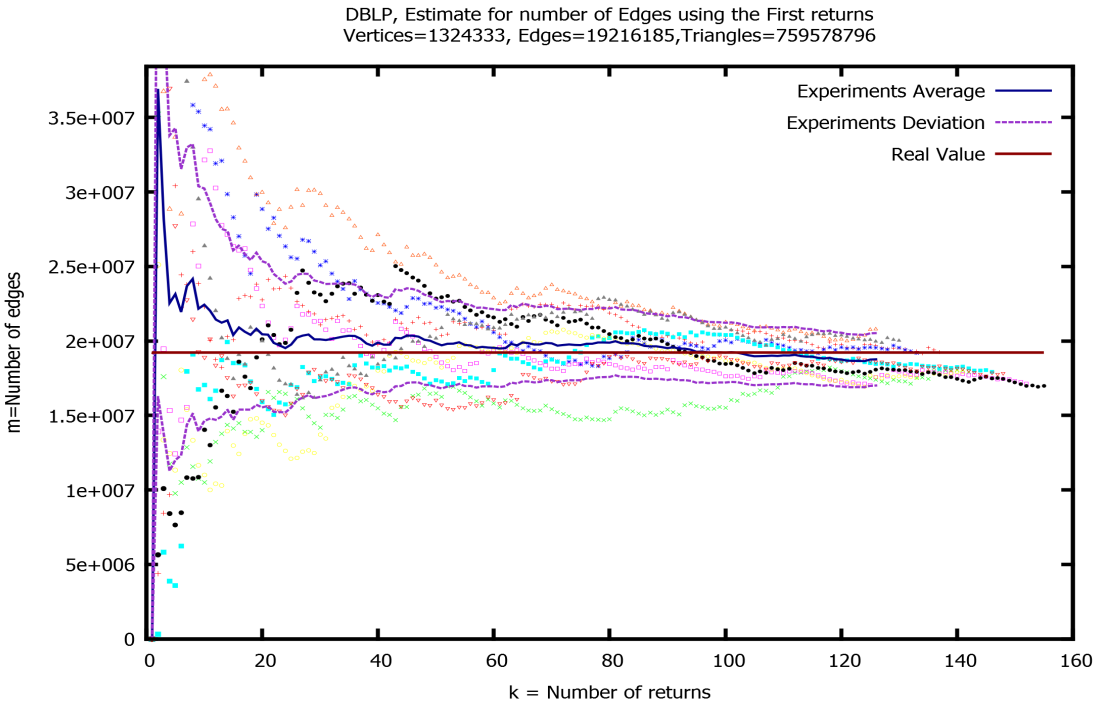


(a) Estimate based on returns to a high weight vertex

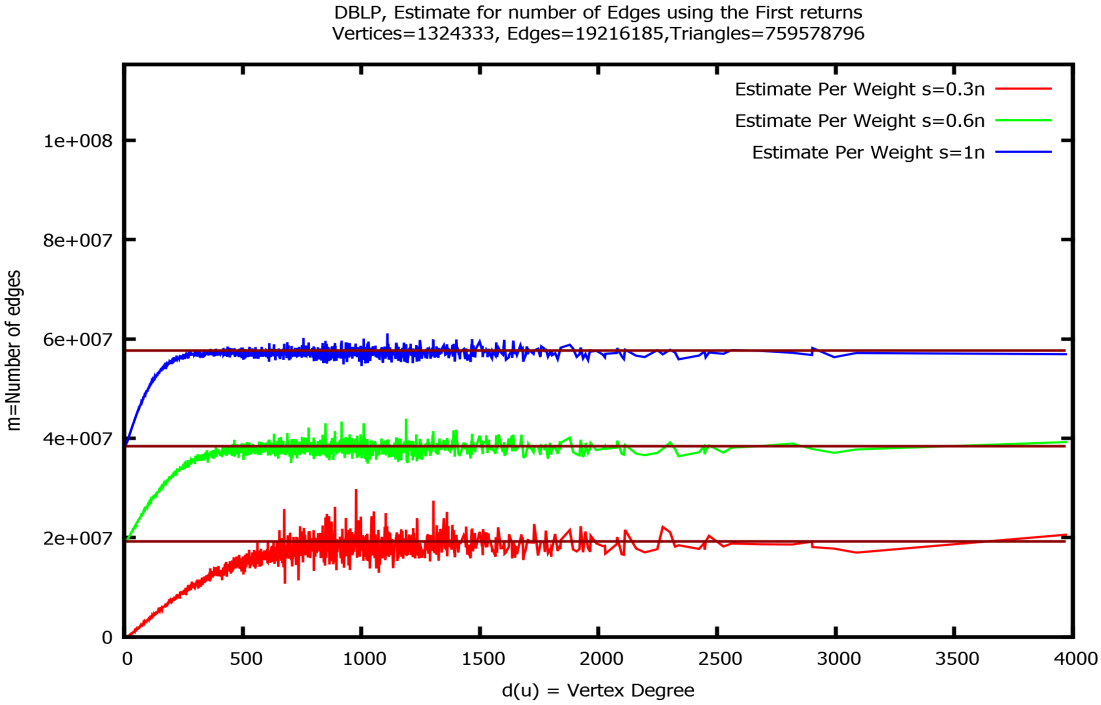


(b) Estimate as a function of vertex weight

Figure 6.23: Vertex estimates for the DBLP co-author graph based on the first return times of a VRW

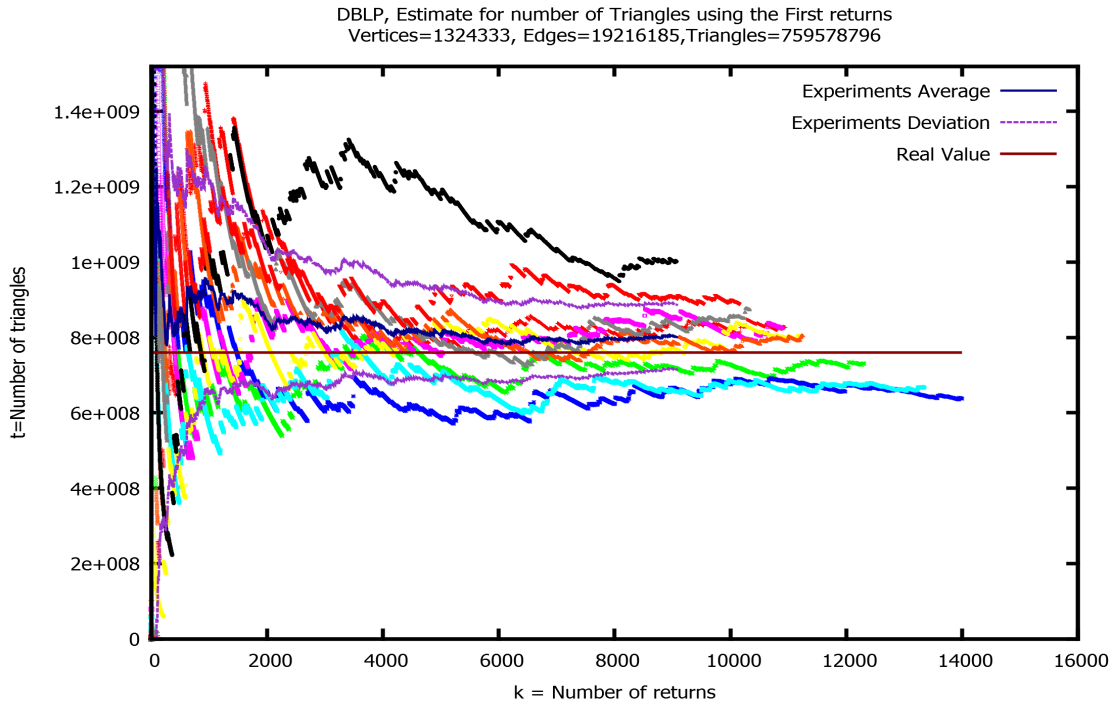


(a) Estimate based on returns to a high weight vertex

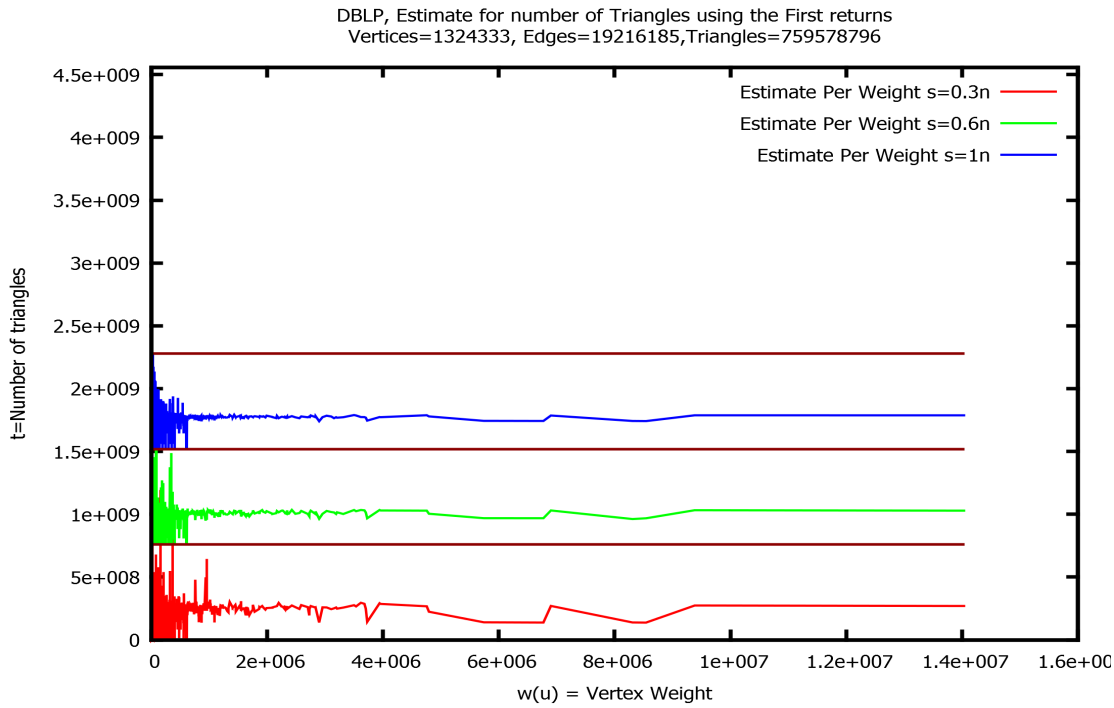


(b) Estimate as a function of vertex weight

Figure 6.24: Edge estimates for the DBLP co-author graph based on the first return times of a SRW

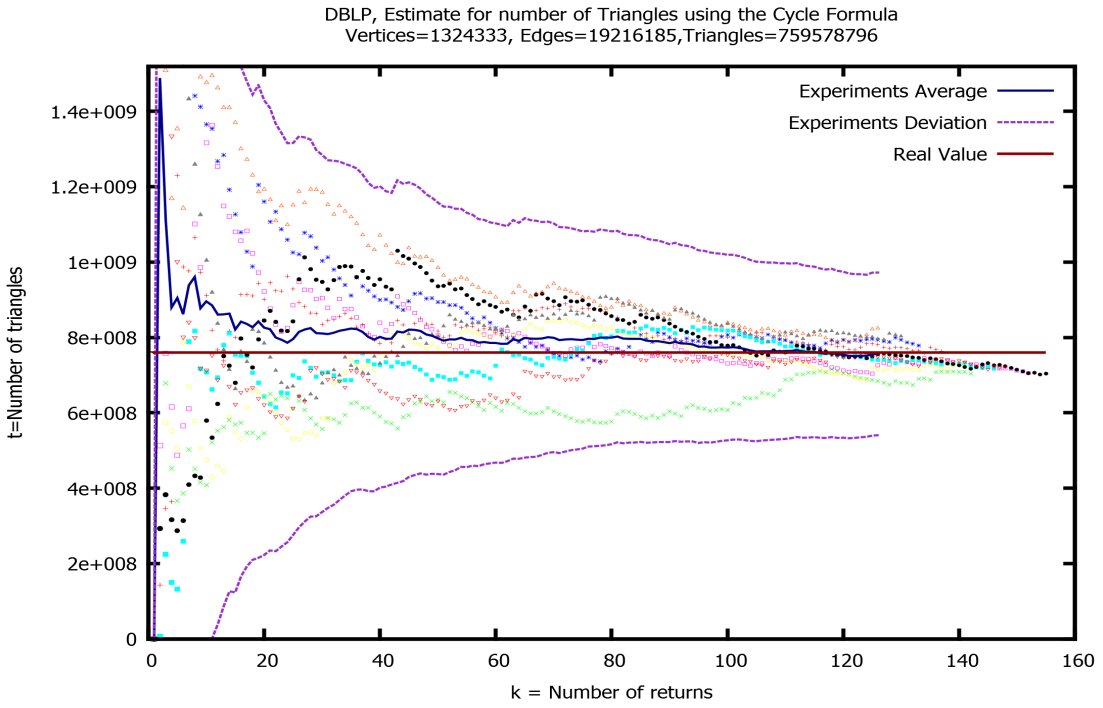


(a) Estimate based on returns to a high weight vertex

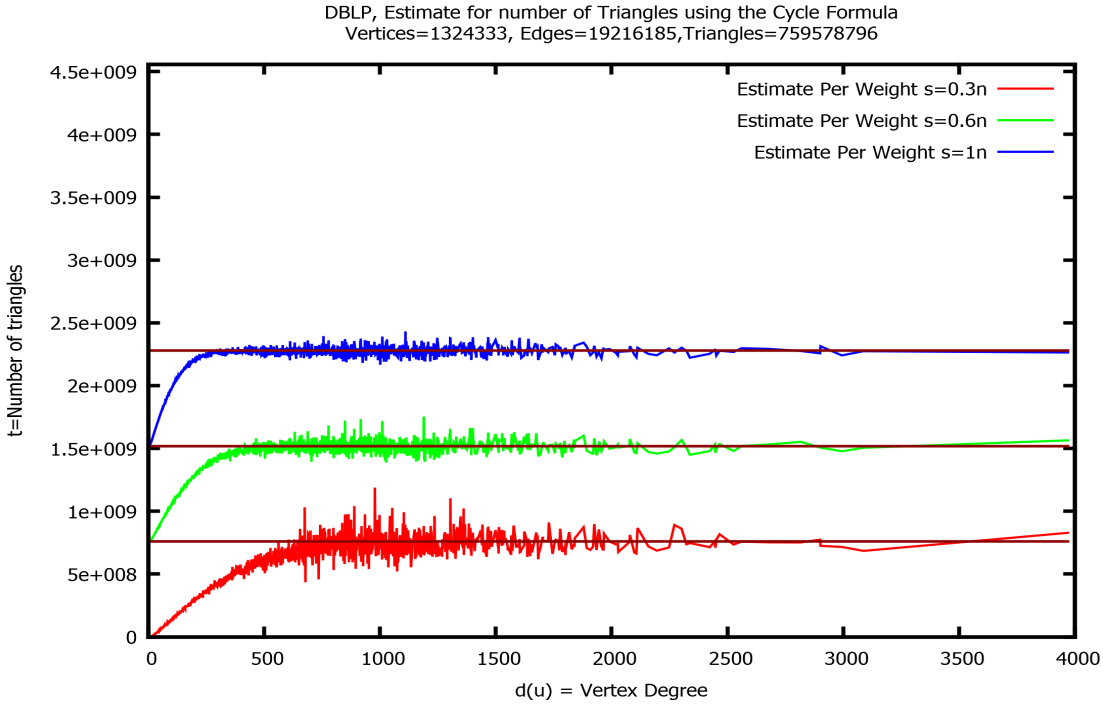


(b) Estimate as a function of vertex weight

Figure 6.25: Triangle estimates for the DBLP co-author graph based on the first return times of a TRW

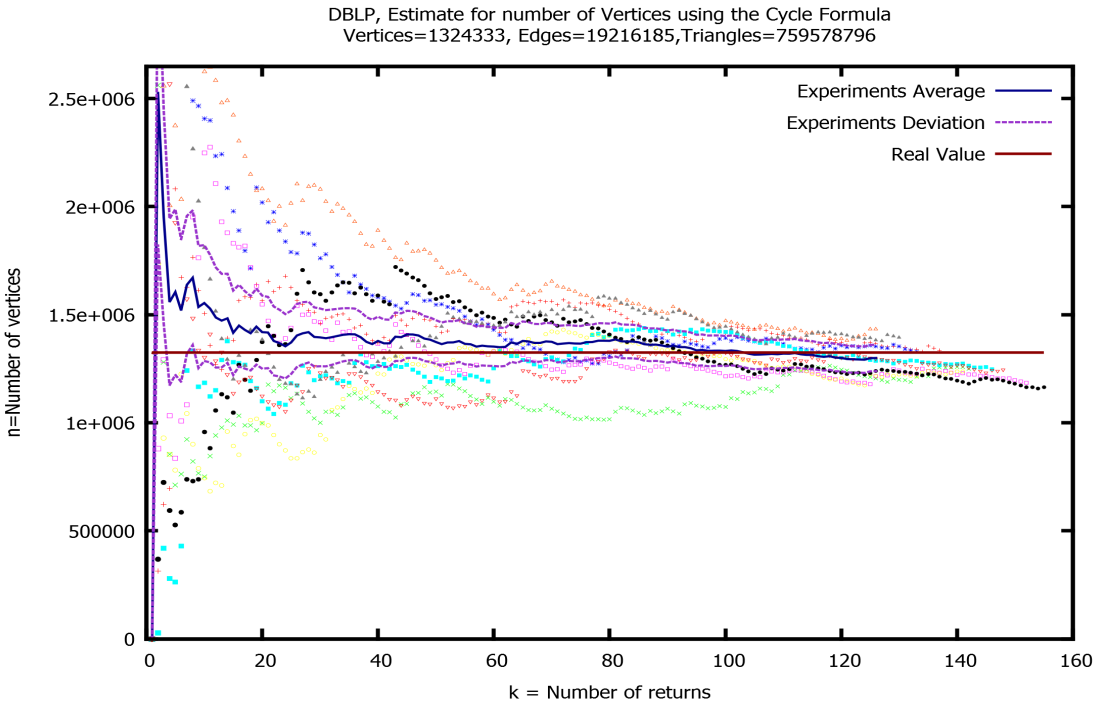


(a) Estimate based on returns to a high weight vertex

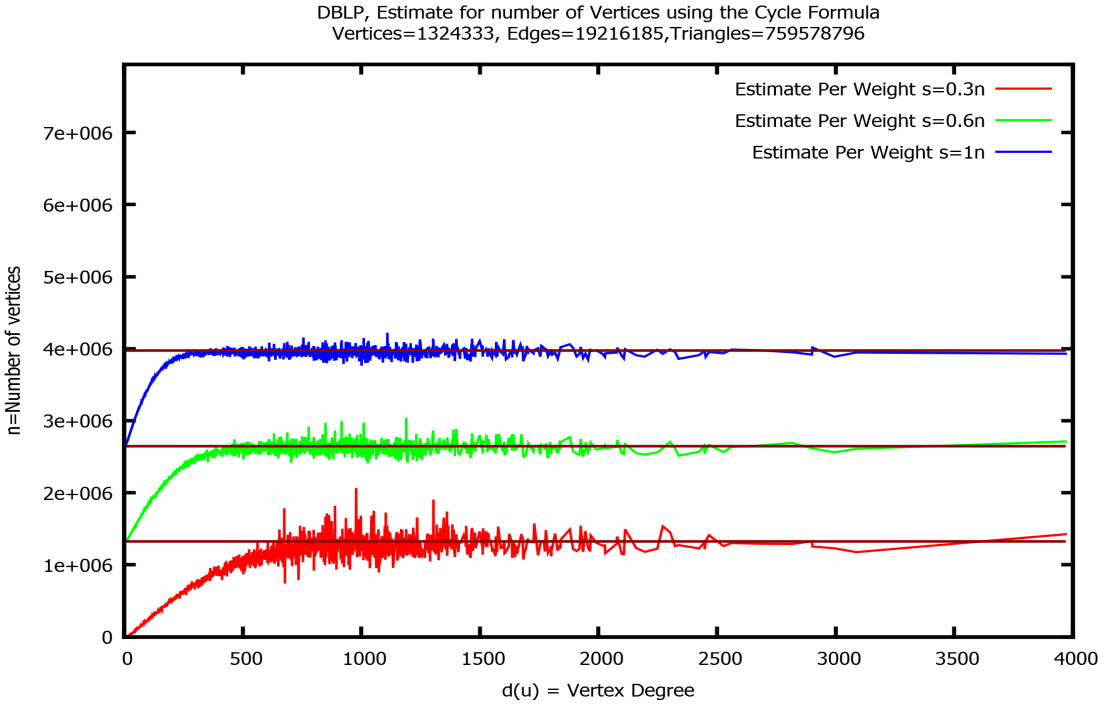


(b) Estimate as a function of vertex weight

Figure 6.26: Triangle estimates for the DBLP co-author graph based on the CFRP



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.27: Vertex estimates for the DBLP co-author graph based on the CFRP

multiple publications with another author then this was expressed through multi-edges. This resulted in a graph consisting of 1324333 vertices, 19216185 edges and 759578796 triangles.

This network is different to other large online networks in the sense that there is a natural limit on the degree each vertex can have, since this corresponds to the number of publications an author can realistically make. This results in an interesting power-law with a rather steep cut-off compared to the number of vertices, i.e. the maximum degree was around 500. Despite this limitation and the fact that the distinction between low degree/weight and high degree/weight vertices is not as clear as other networks, the methods employed still worked surprisingly well.

For this network the estimates converged generally well, slower than in the generated test graphs, faster than in the case of the Google Web graph sample and is comparable to the case of the SlashDot Zoo network.

Estimate of number of vertices. In Figure 6.23 the left hand side figure plots the estimate of the number of vertices over time for the highest weight vertex accessed by the walk. The rate of convergence is slower than in the randomly generated test graphs, however it is significantly faster than the Google web sample. In the right hand side figure we see the vertex estimate $\widehat{n}$ for all sampled vertices, averaged over the vertices of the same weight. In addition, in Figure 6.27 we can see the estimates of the number of vertices based on the CFRP. The underlying walk in this case is a SRW. Specifically it is the same SRW we used to estimate the number of edges with, which would justify the similar accuracy of the two estimates.

Estimate of number of edges. In Figure 6.24 we see the edge estimate $\widehat{m}$ of this graph using a simple random walk. The performance of this walk appears better than the weighted counterpart used above to estimate the number of vertices. It converges slower than the simulated graphs, but faster than the web graph sample of Google.

Estimate of number of triangles. In Figure 6.25 we see the triangle estimate $\widehat{t}$ obtained from the highest weight vertex, using the walk described in Equation (5.22). As

in the previously presented results, the estimates presented here show how the estimate changes based on returns to a single high weight vertex and in addition we group all vertices according to weight. We note how while the maximum weight in this graph is considerably lower than other large online networks, the maximum weight of the TRW is comparable, indicating a number of vertices which are included in a large number if triangles. In general the estimates are reasonable and seem to converge quickly.

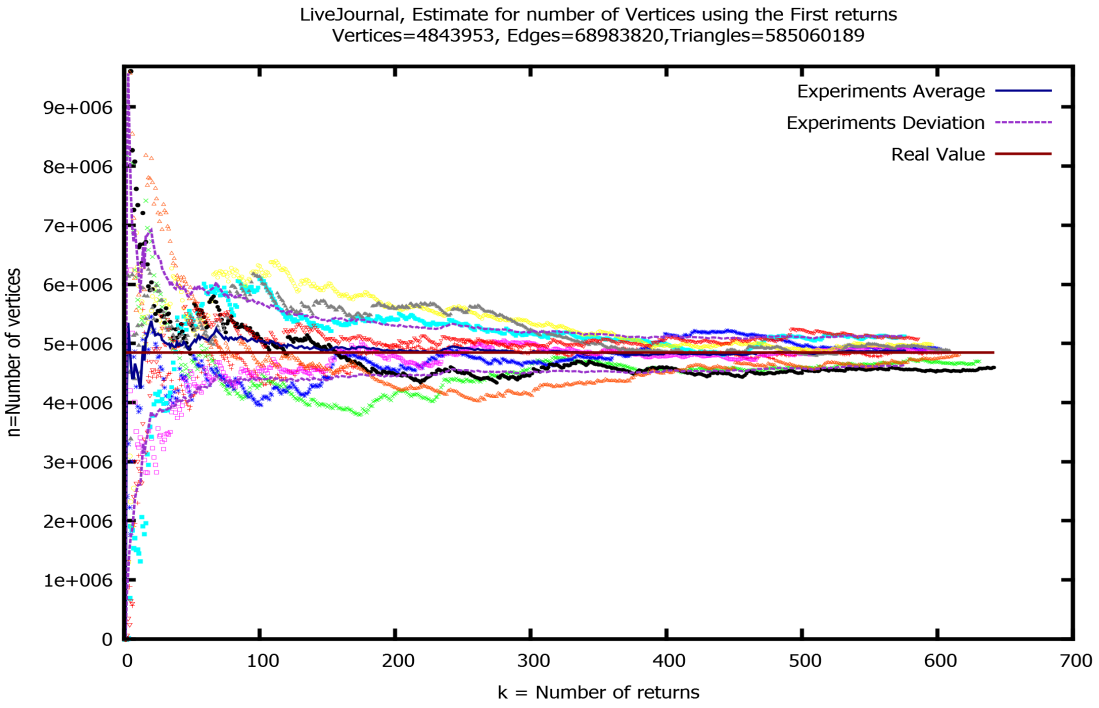
6.1.6 LiveJournal

LiveJournal is a network which allows users to publish content as well as maintain links with other users. The website is self-described as "a community publishing platform, wilfully blurring the lines between blogging and social networking" [94]. It was created on April 15, 1999 by American programmer Brad Fitzpatrick. Originally the creator needed a platform to keep his classmates up to date with his activities, but since then this OLSN has grown and now receives around than 10 million visits per day[5]. The dataset we have used was obtained from [83] and consists of 4843953 vertices, 68983820 edges and 285730264 simple triangles in its largest WCC. However due to multi-edges we have found the number of non-simple triangles to be $5.85 * 10^8$. For this network the estimates converged generally well, slower than in the generated test graphs, but faster than in the case of the Google Web graph sample and comparably to the SlashDot and DBLP networks.

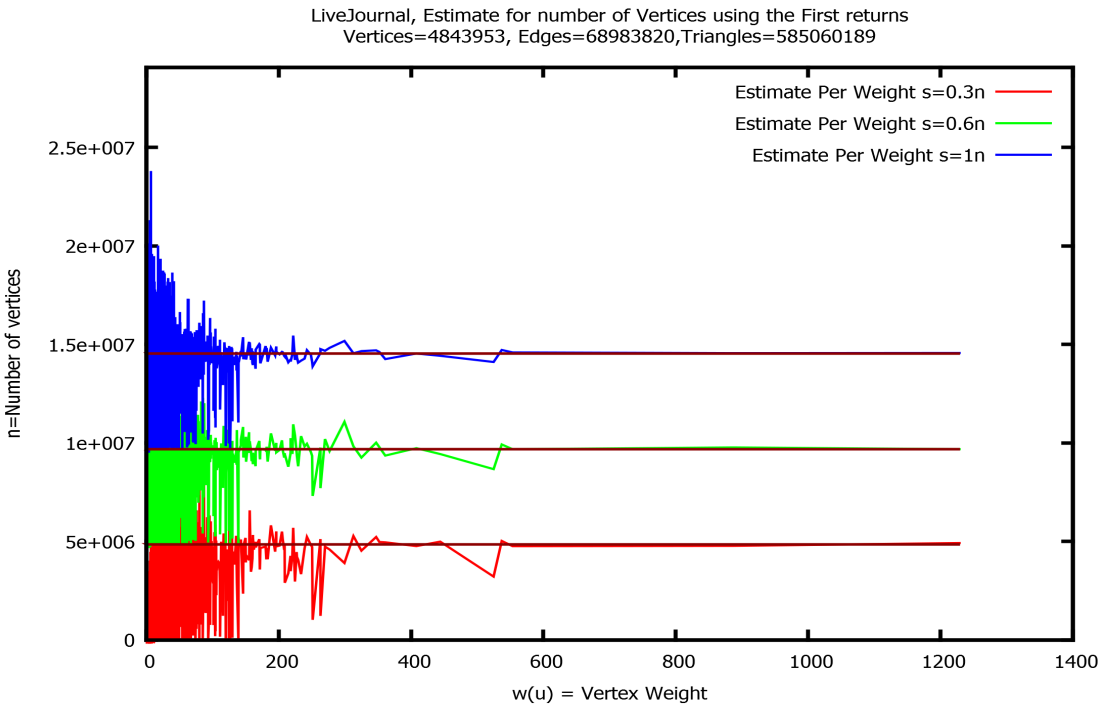
The estimates for this network are organized in the same way as the other networks. The first return time estimators for vertices, edges and triangles can be seen in Figures 6.28,6.29 and 6.30 respectively. In addition the estimates based on the CFRP for vertices and triangles can be seen in Figures 6.32 and 6.31 respectively.

Overall estimation was fairly accurate in all cases, which is a surprising result considering the unique nature of this network. While the network is not an OLSN specifically, the large number of closed triangles indicates there is a strong social aspect to it.

[5]Based on Alexa.com estimates on 14/07/2014

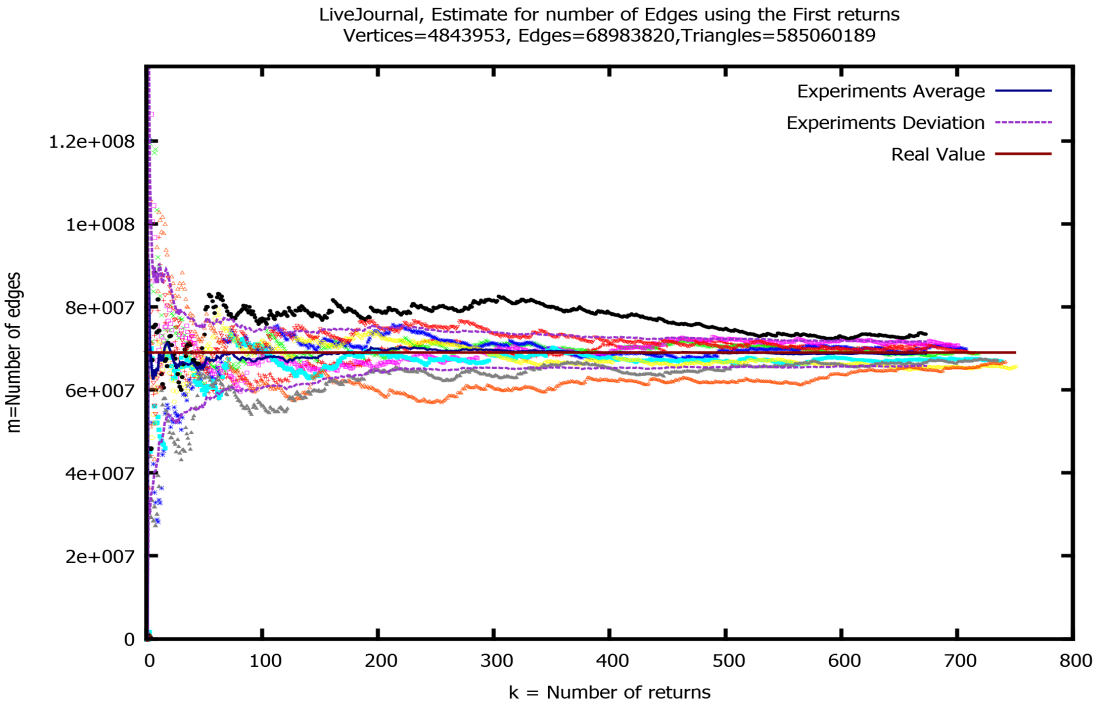


(a) Estimate based on returns to a high weight vertex

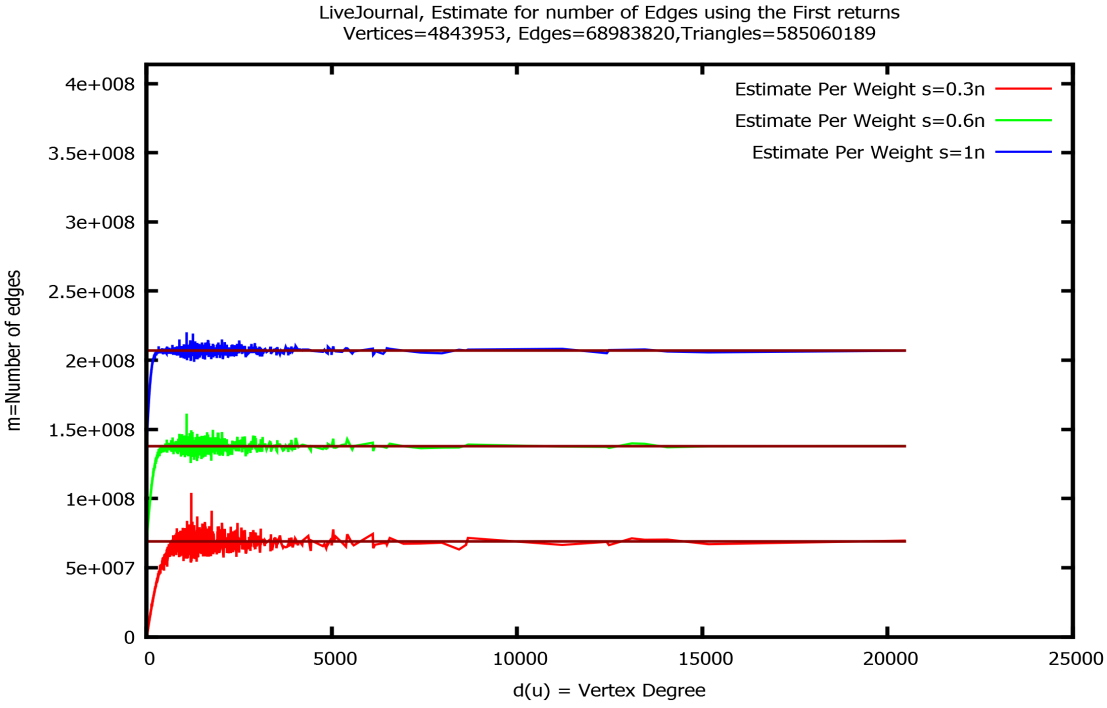


(b) Estimate as a function of vertex weight

Figure 6.28: Vertex estimates for the LiveJournal network based on the first return times of a VRW

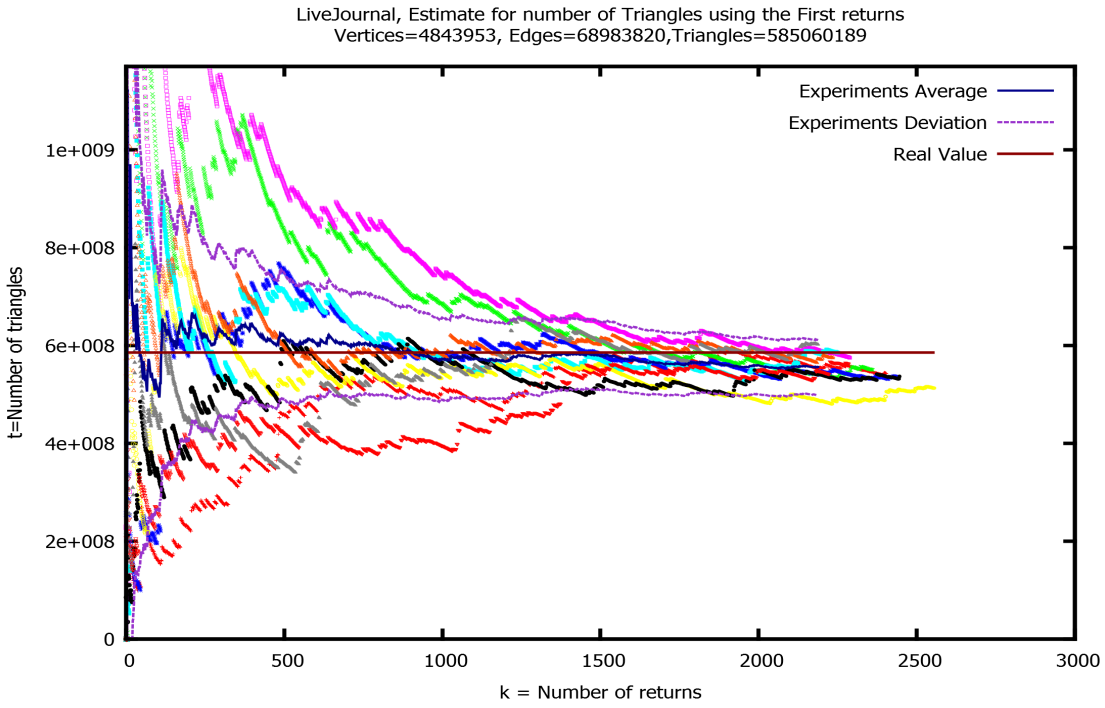


(a) Estimate based on returns to a high weight vertex

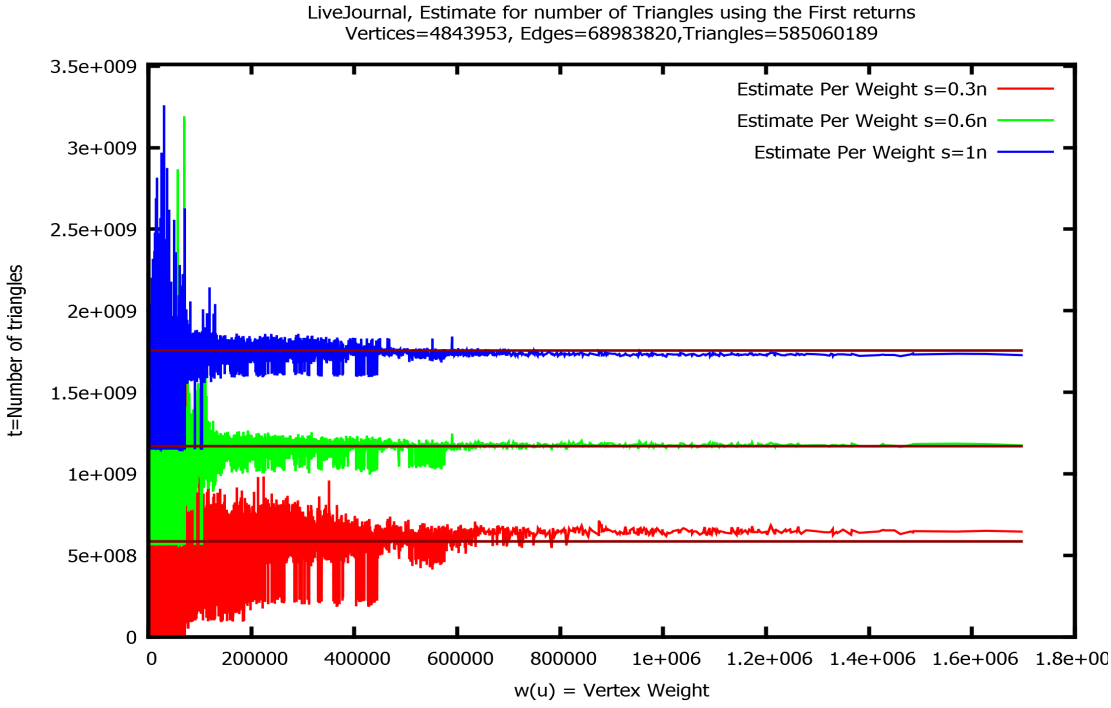


(b) Estimate as a function of vertex weight

Figure 6.29: Edge estimates for the LiveJournal network based on the first return times of a SRW

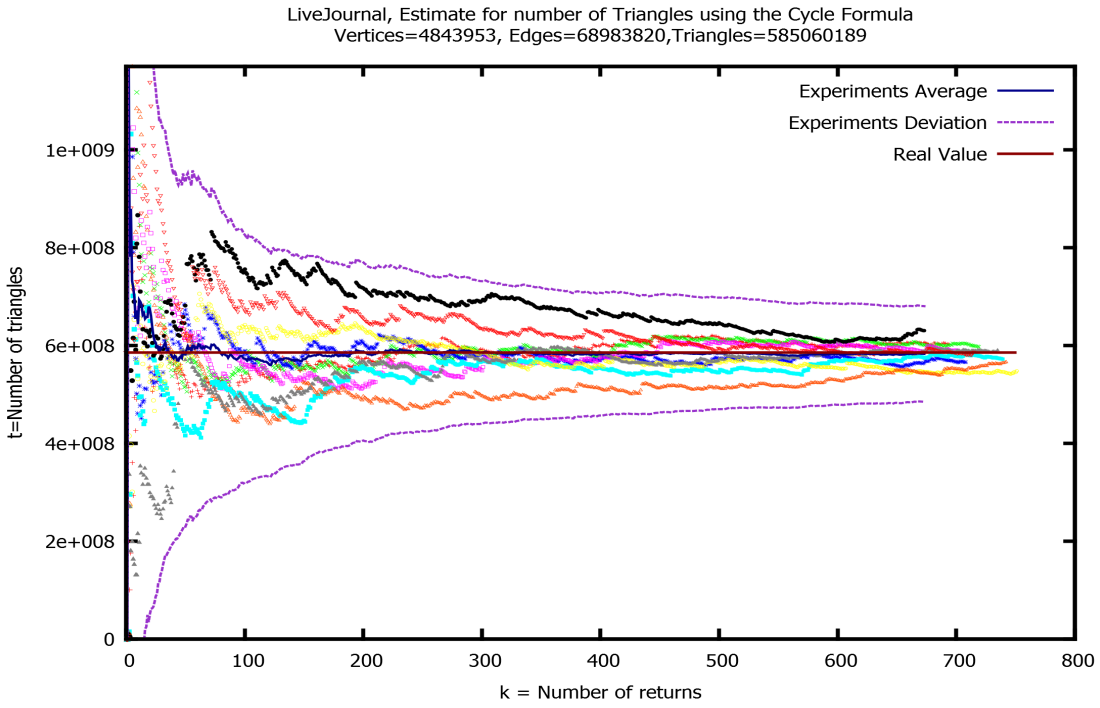


(a) Estimate based on returns to a high weight vertex

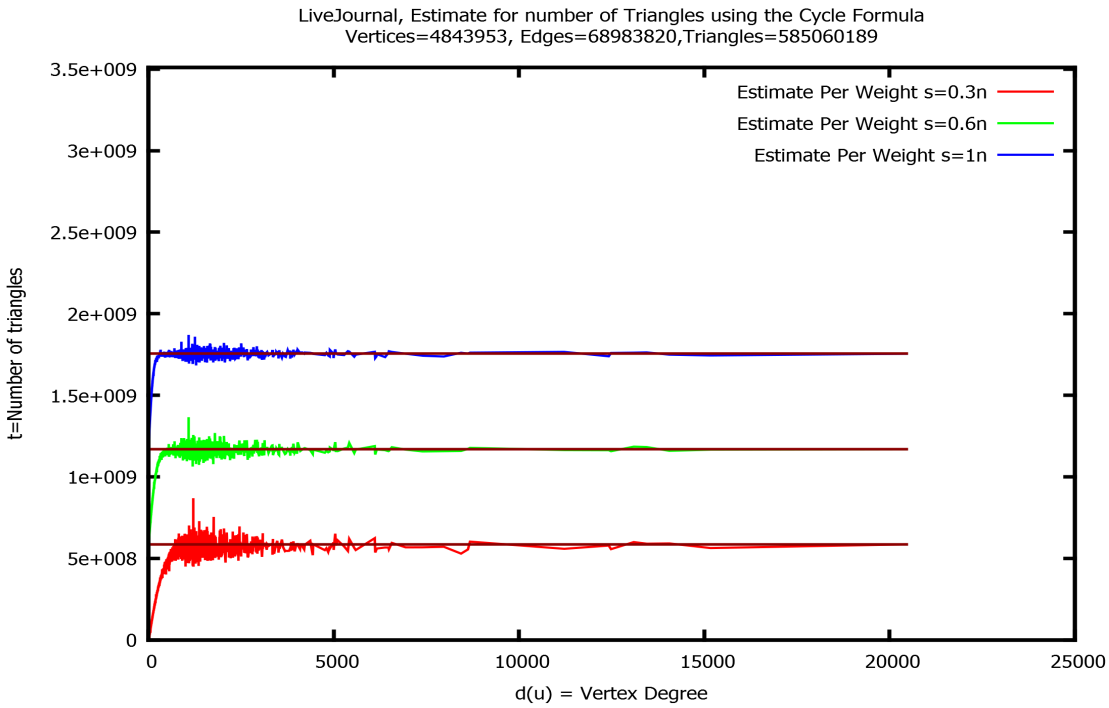


(b) Estimate as a function of vertex weight

Figure 6.30: Triangle estimates for the LiveJournal network based on the first return times of a TRW

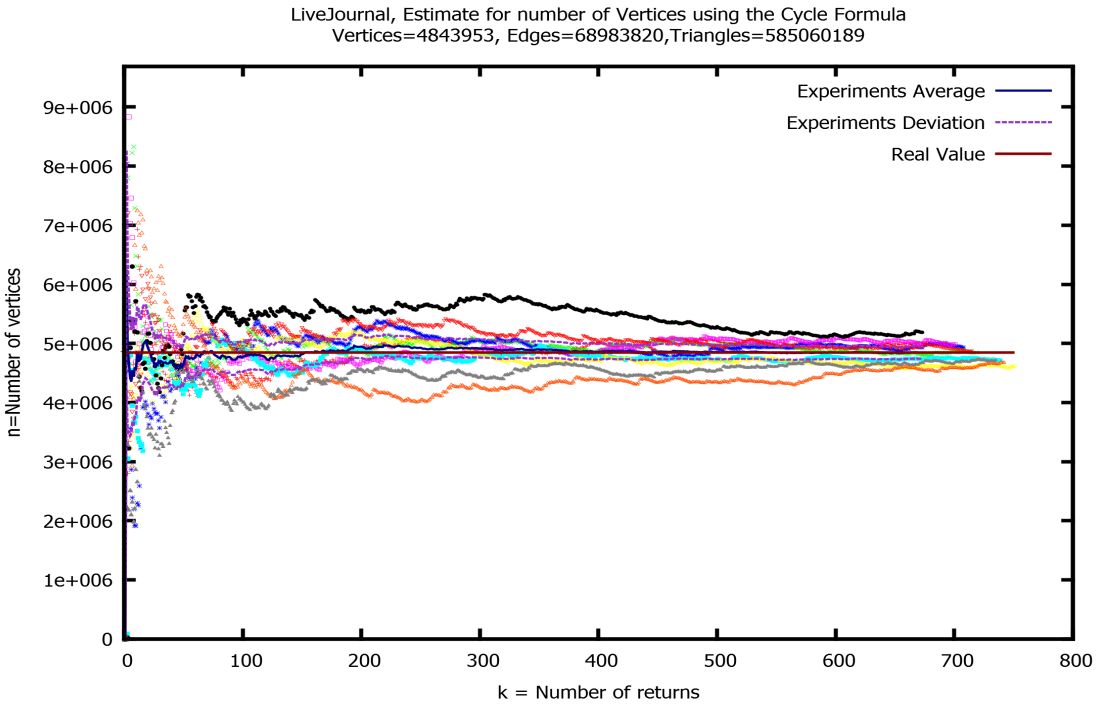


(a) Estimate based on returns to a high weight vertex

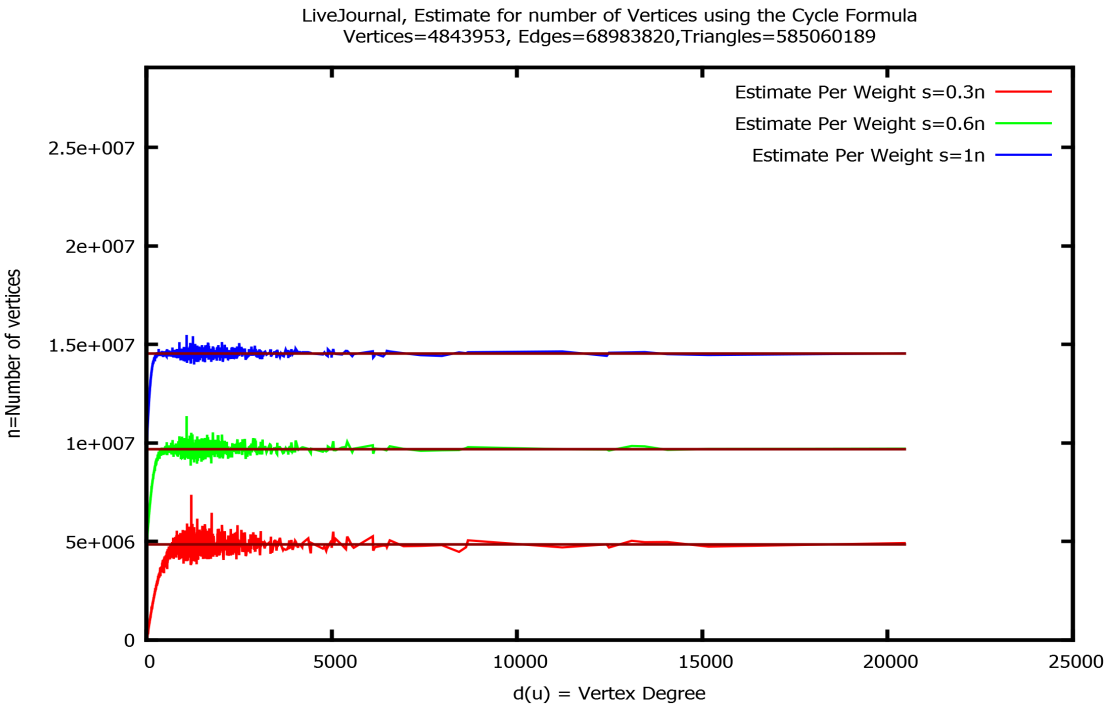


(b) Estimate as a function of vertex weight

Figure 6.31: Triangle estimates for the LiveJournal network based on the CFRP



(a) Estimate based on returns to a high weight vertex



(b) Estimate as a function of vertex weight

Figure 6.32: Vertex estimates for the LiveJournal network based on the CFRP

6.1.7 General Discussion of the results

As we can see from the results, while the methods work reasonably well on all graphs they are deployed on, real graphs such as the Google Web sample show with more fluctuation of the results, this is in part on the structure of the graph. While it is generally accepted that online networks are expanders, they are not such good expanders as random graphs generated using the Barabási/Albert model. However while the data does fluctuate, the tendency to track the real value is obvious, at least in the cases of the VRW and SRW, and in the case of TRW when $s \geq 0.6$.

It is also worth noting that while we choose to stop the walk when the number of steps $s = n$, in fact we only cover part of the graph. This is due to the multiple revisits that many vertices get. This means that in the context of online networks, provided we use appropriate caches, we can run a walk for a high number of steps but the actual number of queries we make to the server would be lower. For example in the case of the Google web sample, after n steps, the SRW had visited around than $2.8 * 10^5$ vertices, less than half the total number of vertices.

However what we should note about e.g. Figure 6.14a is that the number of returns to the highest weight vertex for each of the individual experiments varies greatly. The reason for this is unknown, but presumably it is due to the weights of this walk and the structure of the Google web sample. It may be the case that some walks reached path-like subgraphs from which it was hard to return. The work by A. Z. Broder *et al.* [17] gives a detailed view on the structure of the web and to the existence of such parts (referred to as "tendrils").

6.2 Discussion of results and conclusions

From Equation (5.4) (repeated below)

$$\mathbf{Pr}\left(\left|\frac{Z(k)}{k} - \mathbf{E}T_u^+\right| \geq \epsilon \mathbf{E}T_u^+\right) \leq \frac{1}{\omega},$$

we can see a crude measure of convergence $(1-1/\omega)$ to within $(1\pm\epsilon)\mathbf{E}T_u^+$ of the expected value of return time $\mathbf{E}T_u^+$. The concentration $\epsilon = \sqrt{C\omega/k}$, is based on the assumption that the walk is rapidly mixing (C constant) and that a large enough number, k, of return time samples can be taken to make ϵ small. In order to get k large, and to also keep the run time t of the walk short, we need a vertex u for which $\mathbf{E}T_u^+ = 1/\pi_u$ is small. That is, there exists a vertex u such that $\pi_u \gg \sum_v \pi_v/n$; i.e. there is some vertex u which has a stationary distribution which is much greater than the average. While this is true for power-law degree distribution graphs and SRWs it is not always the case. So we point out that all SRW based methods which are based on the first return times such as e.g. the CFRP (presented in 5.1.2) as well as the edge estimator based on the SRW (Section 5.2.1) would require a significantly larget number of steps on graphs with near uniform stationary distributions, such as in the case of d-regular graphs or Erdős-Rényi random graphs. For graphs with stationary distribution $\pi_v \approx 1/n$ for all vertices, we would have to wait for $\Omega(kn)$ walk steps for a large enough sample, whereas we aim to sample in sub-linear time if at all possible.

However our approach makes use of WRWs to estimate vertices and triangles. If there are high weight vertices and assuming the graph is an expander then the weight bias ensures that we will quickly discover high weight vertices **whp**. Additional details on the bounds of the discovery of such vertices in power-law graphs can be found in [36]. The use of appropriately designed RWs to quickly discover high degree vertices will be further analysed in Chapter 7.

We have presented a general framework for estimating network parameters using appropriately designed random walks. We have shown that the experimental results on manufactured graphs as well as graphs sampled from real network data support the techniques we propose. Moreover our methods give reasonable estimates in a short number of steps. We believe there is great potential on this framework to estimate other properties in graphs, such as degree sequence, variance in degree and measures of expansion.

One particular advantage of our method, is that we can make property estimates even

if the walk is terminated prematurely. For example if the server black listed our queries on the network at any point. Conventional methods such as breadth first search cannot adapt in this way.

These results are encouraging since it is feasible to use these methods on real networks, and we require a minimal amount of resources to do so. In theory a random walk requires no (or little) memory, and the results obtained are reasonably good even for $0.3n$ steps, as we can see from Figures 6.1b,6.2b.

A future study for us will be to run the walk of live networks such as Twitter etc, and see how easily we obtain reasonable estimates. While the elapsed time to run the walk may be large due to query limitations, the overheads are small and it can be left running in background on a single processor, with no significant overheads in program development.

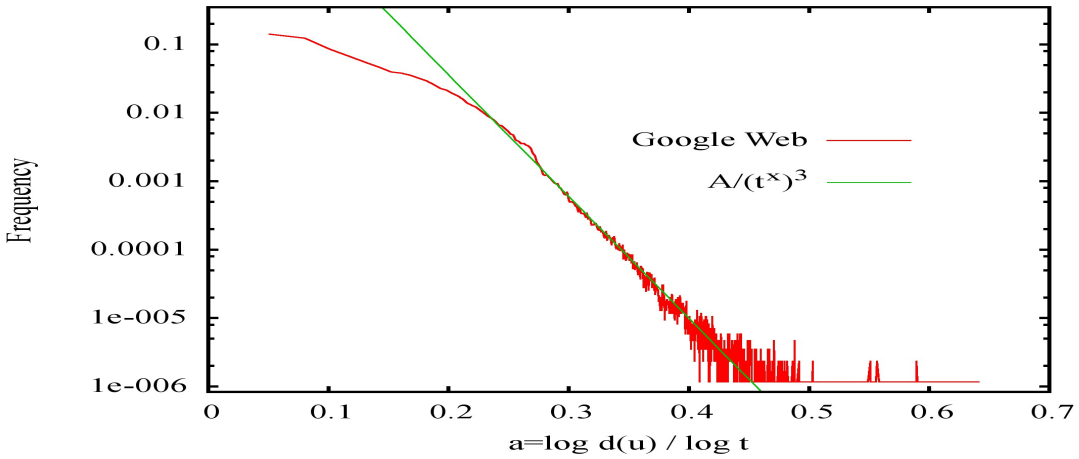
Chapter 7

Fast Discovery of High Degree Vertices

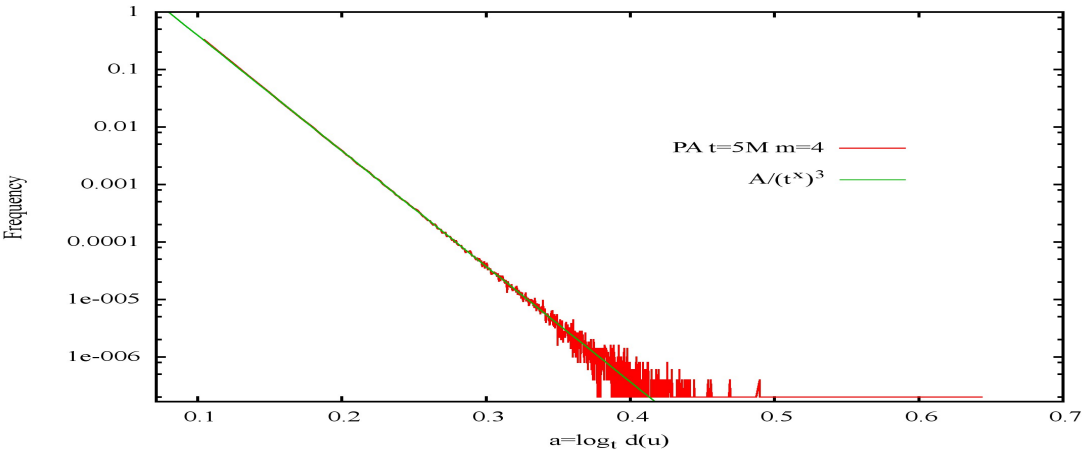
In this chapter we present an approach to discover all high degree vertices in a power-law graph, quickly. The ideas developed are joint work with C. Cooper and T. Radzik and appear in [36, 37].

7.1 Problem motivation

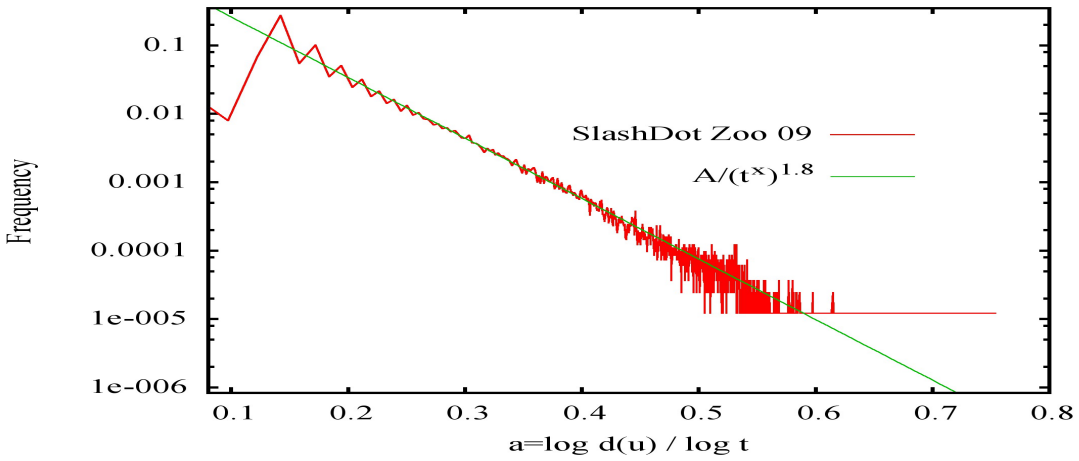
Many large networks have a power-law degree sequence. Thus, although the majority of the vertices have constant degree, a very distinct minority have very large degrees. This particular property is the significant defining feature of such graphs. A log-log plot of the degree sequence breaks naturally into three parts. The lower range (small constant degree) where there may be curvature, as the power law approximation is incorrect (see for example Figure 2.2 in page 47). The middle range, of large but well represented vertex degrees, which give the characteristic straight line log-log plot of the power law coefficient. In the upper tail, where the sequence is far from concentrated, the plot is a spiky mess. See for example Figure 7.1b (the degree sequence of a simulated preferential attachment graph with $m = 4$ edges added at each step), Figure 7.1a (the degree sequence of the underlying graph of a sample of the WWW) or Figure 7.1c (the degree sequence of the underlying graph the SlashDot Zoo Social network). In both cases the x-axis is



(a) Degree distribution of sample of size 8.7×10^5 of G_W, the underlying graph of the WWW



(b) Degree distribution of a realization of $G(c, m, t)$, $c = 3, m = 4, t = 5 \times 10^6$



(c) Degree distribution of underlying graph of a snapshot of SlashDot network

$a = \log d / \log t$, where d is vertex degree, and t is the size of the graph.

In Chapter 4 we presented our work in network sampling and property estimation. We have also shown the importance of starting the random walk based methods from high degree/weight vertices. This can be seen in Equation (5.4) which states that higher degree/weight vertices would result in quicker convergence in the property estimates. This Chapter focuses on methods which we could use to quickly discover a large proportion of these high degree/weight vertices in order to give us flexibility in the choice of the starting vertex.

We focus on sampling higher degree vertices, including those in the middle range as well as the upper tail. Our aim is to find *all these vertices*, and we propose a provably efficient method of obtaining those vertices in sub-linear time using a weighted random walk. One reason for finding all the higher degree vertices is that the upper tail is not concentrated, so no sub-sample will be representative. We consider a weighted random walk because, as there are few vertices even in the middle range, a simple random walk may take too long to obtain a statistically significant sample. Coupled with this is the impression that in many networks, for example the WWW, it is the high degree vertices which are important, both as hubs and authorities, and for page-rank calculations.

Our eventual aim is to devise search algorithms based on weighted random walks which quickly locate all high degree vertices in arbitrary connected graphs with a heavy tailed degree sequence. To do this, we use a random walk with transition probability along edge $\{x, y\}$ proportional to $(d(x)d(y))^b$, where $d(x)$ is the degree of vertex x, and b is a positive constant.

Our work in fast discovery of high degree vertices can be split into two parts. In the first part we make a detailed investigation of the best kind of weighted random walk to find high degree vertices in a graph generated using the preferential attachment model, and with known degree sequence power law c. In this case we find the optimal value $b = b(c)$ to use in the transition probabilities of the weighted random walk. The theory is developed in Sections 7.1.1-7.1.2, and the experimental results given in Section 7.1.3.

Because we use weighted random walks whose mixing properties are not known, the analysis we make for the preferential attachment model does not use notions of mixing time or stationary distribution.

In the second part of this work, in Section 7.2, we discuss what can be obtained in the case of general heavy tailed degree sequences, when the power law c is unknown. Our analysis in Section 7.2 uses notions of mixing time and stationary distribution, and we quantify things as best we can. In particular, we develop reasoning to suggest a weight of $b = \frac{1}{2}$ could be used as a search heuristic, and give experimental evidence on the effectiveness of this choice. Experimentally, setting $b = \frac{1}{2}$ works well as a heuristic in a wide range of real networks irrespective of their power-law coefficient c.

Methods based on RWs are commonly used for graph searching and crawling. Stutzbach *et al.* [123] compare the performance of breadth first search (BFS) with a simple random walk and a Metropolis Hastings random walk on various classes of random graphs as a basis for sampling the degree distribution of the underlying networks. The purpose of the investigation was to sample from dynamic P2P networks. In a related study Gjoka *et al.* [57] made extensive use of these methods to collect a sample of Facebook users. As simple random walks are degree biased they used a re-weighting technique to unbias the sampled degree sequence output by the random walk. This is referred to as a re-weighted random walk in [57]. In both the above cases it was shown the bias could be removed dynamically by using a suitable Metropolis-Hastings random walk.

7.1.1 Preferential attachment graphs, model and search methods

A simple way to generate a graph with a power law degree sequence is to use the preferential attachment method described by Albert and Barabási [4] and shown in Section 2.2.2. We remind that the power-law coefficient c for preferential attachment graphs and web-graphs is given in Equation (2.1), i.e. $c = 1 + 1/\eta$, where η is the expected proportion of edge end points added preferentially (see [26] and Section 2.2.2). As mentioned in Section 2.2.2, we will refer to a graph of size t with expected power-law

coefficient c and a constant number of edges added per step as $G(c, m, t)$ where m is the number of edges added per step.

The parameter η in (2.1) occurs in process models, in the expression for the expected degree of a vertex. Let $d(s,t)$ denote the degree at step t of the vertex v_s added at step s. The expected value of $d(s,t)$ is given by Equation (2.17) repeated below:

$$\mathbf{E}d(s,t) \sim m\left(\frac{t}{s}\right)^{\eta}$$

To generalize this, we consider multi-graphs $G(t)$ on t vertices with properties (i) and (ii) below, where $\epsilon > 0$ and $0 < \eta < 1$ are parameters. We call such graphs *pseudo-preferential*.

(i) When the vertices are relabelled $s = 1, ..., t$ by sorting on vertex degree in descending order, $G(t)$ has a degree sequence which satisfies for each $s \geq 1$,

$$\left(\frac{t}{s}\right)^{\eta(1-\epsilon)} \leq d(s) \leq \left(\frac{t}{s}\right)^{\eta} \log^2 t. \tag{7.1}$$

(ii) For all vertices s in the sorted order, s has at most m edges to vertices $\sigma \leq s$.

Our particular aim is, given $a > 0$, to find all vertices $v \in V(t)$ of degree $d(v) \geq t^a$. Denote by S_a the set of vertices of $G(t)$ of degree $d(v) \geq t^a$. For the following reason, we will assume $a < \eta$. The maximum degree in (7.1) is $\widetilde{O}(t^{\eta})$, and, from (2.17), this is also the maximum expected degree in preferential attachment graphs ($\eta = 1/2$) and web-graphs ($0 < \eta < 1$). We use the notation $\widetilde{O}(f(t))$ as shorthand for $O(f(t)\log^k t)$ where t is the size of $G(t)$ and k is a positive constant.

When we apply a random walk to find all vertices in a set S, then we say that the walk is *seeded* if it starts from some vertex of S. In the context of searching networks such as Facebook, Twitter or the WWW it is not unreasonable to suppose we know *some* high degree vertex without supposing we know all of them. Experimentally, we found the *seeding* condition was not necessary, but a general analysis without this condition would require notions of mixing time and stationarity which our analysis avoids.

The maximum degree of $G(c,m,t)$ is $\widetilde{O}(t^{\eta})$ **whp**, where $\eta = 1/(c-1)$ which explains the bound on a given above. Using this, a t^{1-2ab} run time can be re-packaged as follows. Let $a = \theta\eta$ for $0 < \theta < 1$, then $2ab = \theta(1 - 1/(2c-3))$.

For a graph $G(c,m,t)$ generated by the web-graph process (see Section 2.2.2) the actual value of $d(s,t)$ is not concentrated around $\mathbf{E}d(s,t)$ in the lower tail, but a version of Lemma 7.1 proved in [26] is adequate for our analysis.

In reality the degree sequence (7.1) of graph $G(t)$ is unknown, but η can be estimated as $\eta = 1/(c-1)$ from the power law c of the degree sequence, if this is known. Optimistically setting $\epsilon = 0$ gives a value b for the search algorithm. It's also fair to say that, experimentally, we found putting $b = 1/2$ in the biased random walk was effective for a variety of real networks with a power law degree sequence.

The preferential attachment model, and its generalization, the web-graph model satisfy (7.1) for suitable choices of ϵ, η, m.

Lemma 7.1. *Given $G(c,m,t)$ and a, ϵ and suppose $m > (1/\epsilon)(1/a - 1/\eta)$, where η as in (2.1).*
With high probability for all vertices s, such that $\mathbf{E}d(s,t) \geq t^a$, we have that $d(s,t) \geq \left(\frac{t}{s}\right)^{\eta(1-\epsilon)}$. For all $s \geq \log^2 t$, $d(s,t) \leq \left(\frac{t}{s}\right)^{\eta}\log^2 t$. For all $s \geq 1$, $d(s,t) = \widetilde{O}(t^{\eta})$.

The upshot of this is that all vertices added after step $v = s\log^{2/\eta+1} t$ have degree $d(v,t) = o\left((t/s)^{\eta}\right)$ **whp** since $\mathbf{E}d(u,t) = 2ms\left(\frac{t}{s}\right)^{\eta}$. This observation forms the basis of our sub-linear algorithm.

Proof The upper bound on $d(s,t)$ is given in [26], as is the following degree distribution. For $m \geq 2$, the distribution of $d(s,t)$ is given by

$$\mathbf{Pr}(d(s,t) = m + \ell \mid d(s,s) = m) \leq C\binom{m+\ell-1}{\ell}\left(\frac{s}{t}\right)^{\eta m}\left(1 - \left(\frac{s}{t}\right)^{\eta}\right)^{\ell}.$$

Thus, crudely

$$\mathbf{Pr}(d(s,t) \leq \ell) \leq C\ell^m\left(\frac{s}{t}\right)^{\eta m}.$$

Inserting $\ell = \left(\frac{t}{s}\right)^{\eta(1-\epsilon)}$, and choosing $s = t^{1-a/\eta}$, we find the expected number of vertices $1 \leq v \leq s$ not satisfying the lower bound is of order

$$s\left(\frac{s}{t}\right)^{m\epsilon\eta} = t^{(1-a/\eta)(1+m\epsilon\eta)} = o(1),$$

provided

$$m > \frac{1}{\epsilon}\left(\frac{1}{a} - \frac{1}{\eta}\right).$$

□

We also need lower tail concentration for large sets of vertices.

Lemma 7.2. *Let $d([s],t)$ denote degree of $[s] = \{1, ..., s\}$ at step t. Let $K > 1$. Then*

$$\mathbf{Pr}\left(d([s],t) \leq \frac{2ms}{K}\left(\frac{t}{s}\right)^{\eta}\right) = O(s^{-mK}).$$

Proof We give the proof for $\eta = 1/2$ (preferential attachment), the general proof is similar.

Let $Z_t = d([s],t)$. Then $Z_s = 2ms$, as m edges are added in the first s steps. From step $t \geq s+1$, $Z_t = X_t + Z_{t-1}$, where $X_t \sim Bin(m, Z_{t-1}/(2m(t-1)))$. This comes from the fact that when we create a new edge e in the preferential attachment model, assuming we add a new vertex u at step t, the probability that we select a vertex in s preferentially is $P(e \text{ picks } [s]) = \frac{d([s])}{2m(t-1)} = \frac{Z_{t-1}}{2m(t-1)}$. Since we add m edges at each step, we have m such *i.i.d.* chances of selecting a vertex in s, which is equivalent to a binomial distribution with parameters $p = \frac{Z_{t-1}}{2mt}$ and m.

It follows that $\mathbf{E}Z_t \sim 2ms(t/s)^{1/2}$.

Given $h, c_t, A > 0$,

$$\mathbf{Pr}(Z_t < A) = \mathbf{Pr}(e^{-hZ_t/c_t} > e^{-hA/c_t}).$$

Let $p = \frac{Z_{t-1}}{(2m(t-1))}$, then

$$\mathbf{E}(e^{-hX_t/c_t}) = (1 - p + pe^{-\frac{h}{c_t}})^m$$

since $1 - x \leq e^{-x}$ we have:

$$\begin{aligned}(1 - p + pe^{-\frac{h}{c_t}})^m &= (1 - p(1 - e^{-\frac{h}{c_t}}))^m \\ &\leq (e^{-p(1-e^{-\frac{h}{c_t}})})^m\end{aligned}$$

by using $e^{-x} \leq 1 - x + x^2$ we have:

$$1 - e^{-h/c_t} \geq 1 - (1 - \frac{h}{c_t} + \frac{h}{c_t}^2) = \frac{h}{c_t} - \frac{h}{c_t}^2$$

therefore

$$\begin{aligned}(e^{-p(1-e^{-h/c_t})})^m &\leq e^{-pm(\frac{h}{c_t} - \frac{h}{c_t}^2)} \\ &= e^{-\frac{Z_{t-1}}{(2(t-1))}(\frac{h}{c_t} - \frac{h}{c_t}^2)} \\ &= e^{-\frac{h}{c_t}(1-\frac{h}{c_t})\frac{Z_{t-1}}{2(t-1)}},\end{aligned}$$

Let $c_s = 1, c_t = (1+1/(2(t-1)))c_{t-1}$ so that $c_t \sim (t/s)^{1/2} \sim \mathbf{E}Z_t/(2ms)$. We will choose $h = o(1)$ (see below). Iterating the expression $Z_t = X_t + Z_{t-1}$, and conditioning on the value of Z_{t-1} gives

$$\mathbf{E}(e^{-hZ_t/c_t} \mid Z_{t-1}) \leq \tag{7.2}$$

$$e^{-h\frac{Z_{t-1}}{c_{t-1}}\frac{1+(1-h/c_t)/(2(t-1))}{1+1/(2(t-1))}} \tag{7.3}$$

$$= e^{-h'Z_{t-1}/c_{t-1}}, \tag{7.4}$$

where

$$h(1 - O(h/tc_t)) \leq h' \leq h,$$

and

$$\mathbf{E}(e^{-hZ_s/c_s}) = e^{-h2ms}.$$

The conditioning in (7.2)-(7.4) can be recursed backwards by taking the expectation of Z_{t-1} conditional on Z_{t-2} etc. until we arrive at the expectation of Z_s which is constant. Thus

$$\mathbf{E}(e^{-hZ_t/c_t}) \leq \mathbf{E}\left(e^{-h\frac{Z_s}{c_s}\prod_{j=s}^{t-1}(1-O(h/(jc_j)))}\right) = e^{-h2ms(1-O(h))}.$$

Applying the Markov inequality that $\mathbf{Pr}(Y \geq A) \leq \mathbf{E}(Y)/A$ with $Y = e^{-hZ_t/c_t}$ and $A = e^{-h\mathbf{E}(Z_t)/(Kc_t)}$ where $K > 0$ (from statement of lemma) we have

$$\begin{aligned}\mathbf{Pr}(Z_t \leq \mathbf{E}Z_t/K) &= \mathbf{Pr}\left(e^{-hZ_t/c_t} \geq e^{-h\mathbf{E}(Z_t)/(Kc_t)}\right) \leq e^{-h2ms(1-1/K-O(h))} \\ &= O(s^{-mK}),\end{aligned}$$

on choosing $h = (K \log s)/s = o(1)$. □

7.1.2 Fast high degree vertex discovery Random Walk

Let $d(v) = d(v,t)$ be the degree of vertex $v \in G(t)$, and let $N(v)$ denote the neighbours of v in this graph. The basis of our algorithm is a degree-biased random walk, with transition probability $p(u,v)$ given by

$$p(u,v) = \frac{(d(v))^b}{\sum_{w \in N(u)} (d(w))^b}, \tag{7.5}$$

where $b > 0$ constant. The value of b we will choose in our proofs is optimized to depend on η. Thus for Theorem 7.3, using (2.1), the value of b can be expressed directly as a function of the degree sequence power law c.

The easiest way to reason about biased random walks, is to give each edge e a weight $w(e)$, so that transitions along edges are made proportional to this weight. In the case above the weight of the edge $e = (u,v)$ is given by $w(e) = (d(u)d(v))^b$ so that the transition probability (7.5) is now written as

$$p(u,v) = \frac{(d(u)d(v))^b}{\sum_{w \in N(u)} (d(u)d(w))^b}. \tag{7.6}$$

The inspiration for the degree biased walk above, comes from the β-walks of Ikeda, Kubo, Okumoto and Yamashita [66] which use an edge weight $w(x,y) = 1/(d(x)d(y))^\beta$ to favour low degree vertices. When $\beta = 1/2$ this gives an improved worst case bound of $O(n^2 \log n)$ for the cover time of connected n-vertex graphs.

Some facts about weighted random walks, can be found in Aldous and Fill [2] or Lovasz [95] and are also thoroughly explained in Chapter 3. We make use of the electrical network paradigm where the weight $w(e)$ of an edge e has the meaning of conductance in electrical networks, and the resistance $r(e)$ of e is given by $r(e) = 1/w(e)$. The commute time for a weighted walk is given by Equation (3.29) i.e. $K(u,v) = w(G)R_{\mathit{eff}}(u,v)$. Where $w(G) = 2\sum_{e \in E} w(e)$ and $R_{\mathit{eff}}(u,v)$ is the effective resistance between u and v, when G is taken as an electrical network with edge e having resistance $r(e)$. For a random walk starting in a set S, the cover time of S satisfies the Matthews of Equation (3.30), repeated below:

$$C_S^* \leq \max_{u,v \in S} H(u,v) \log |S|.$$

We next give a general result for web-graphs $G(c, m, t)$, which is also valid for related models such as scale-free graphs. For the class of graphs $G(c, m, t)$, the lower bound on the degree of vertex s becomes less concentrated as s tends to t, so that the value of ϵ we must choose for our lower bound in (7.1) increases with s. Thus, as the vertex degree t^a decreases, the upper bound on the algorithm runtime increases in a way which depends on a, c, m. As long as we incorporate this dependence, Theorem 7.3 says that if we search $G(c, m, t)$ using a random walk with a bias b proportional to the power law c then, (i) we can find all high degree vertices quickly, and (ii) the time to discover all vertices is of about the same order as for a simple random walk.

Theorem 7.1. *By choosing*

$$b = \frac{1-\eta}{\eta(2-\eta(1-\epsilon'))},$$

where $\epsilon' = \epsilon$ for the pseudo-preferential graphs (Theorem 7.2), and $\epsilon' = 0$ for the web graphs (Theorem 7.3), it follows that $w(G) = \widetilde{O}(t)$.

Proof The proofs for the pseudo-preferential graphs and the web-graphs are very similar, so we present them together. For the web-graphs, all statements should be understood as "with high probability".

We define a graph G^* on vertices $1, 2, \ldots, t$ which has the same degree sequence as graph G, and is built in a similar iterative process: for each $v = t_0, t_0 + 1, \ldots, t$, add m edges from vertex v to some earlier vertices. In graph G, edges are selected according to a random preferential process, while in graph G^* according to the deterministic process which greedily fills the in-degrees of vertices, giving preference to the older vertices. In both graphs, if (x, y) is a directed edge, then $y < x$ (the edges point from x towards the earlier vertex y).

Assume $b > 0$ and define

$$\begin{aligned} \bar{d}(v) &= \left(\frac{t}{v}\right)^{\eta}, \\ \bar{w}(G) &= 2 \sum_{\{x,y\} \in E(G)} \left(\bar{d}(x)\bar{d}(y)\right)^{b}. \end{aligned}$$

Using Equation (7.1) for the pseudo-preferential graphs and Lemma 7.1 for the web-graphs, we see that in both graphs, the degree of vertex v is

$$d(v) = \bar{d}(v) \cdot O(\text{polylog}(t)), \tag{7.7}$$

so we have

$$w(G) = \bar{w}(G) \cdot O(\text{polylog}(t)). \tag{7.8}$$

Assume graph G^* obtained from G by repeatedly swapping edges. Whenever there is a pair of edges (x, y), (u, v) such that $x < u$ but $y > v$, then replace them with edges (x, v) and (u, y). If $A > B$ and $C > D$ then $(A - B)(C - D) > 0$ so $AC + BD > AD + BC$. Thus each swap increases $\bar{w}(G)$ because

$$(\bar{d}(x))^b > (\bar{d}(u))^b \quad \text{and} \quad (\bar{d}(y))^b < (\bar{d}(v))^b$$

implies

$$(\bar{d}(x))^b(\bar{d}(v))^b + (\bar{d}(u))^b(\bar{d}(y))^b \;>\; (\bar{d}(x))^b(\bar{d}(y))^b + (\bar{d}(u))^b(\bar{d}(v))^b.$$

Therefore,

$$\bar{w}(G^*) \geq \bar{w}(G). \tag{7.9}$$

By construction, a vertex v in G^* has incoming edges originating from consecutive vertices $\text{first}(v), \text{first}(v) + 1, \ldots, \text{last}(v)$. Thus we have

$$\begin{aligned}
\bar{w}(G^*) &= 2 \sum_{\{y,x\} \in E(G^*)} \left(\bar{d}(x)\bar{d}(y)\right)^b \\
&= 2 \sum_{x=1}^{t} \sum_{y=\text{first}(x)}^{\text{last}(x)} \left(\bar{d}(x)\bar{d}(y)\right)^b \\
&\leq 2 \sum_{x=1}^{t} d(x) \left(\bar{d}(x)\bar{d}(\text{first}(x))\right)^b \\
&\leq O(\text{polylog}(t)) \cdot \sum_{x=1}^{t} \left(\bar{d}(x)\right)^{1+b} \left(\bar{d}(\text{first}(x))\right)^b .
\end{aligned} \tag{7.10}$$

This follows from the definition in Equation (7.7). Now we bound first(x). There are at most $m \cdot \text{first}(x)$ edges outgoing from vertices $1, 2, \ldots, \text{first}(x)$, and these edges fully fill the in-degrees of vertices $1, 2, \ldots, x-1$, so

$$m \cdot \text{first}(x) \;\geq\; \sum_{z=1}^{x-1} d_{in}(z) \;=\; \sum_{z=1}^{x-1} (d(z) - d_{out}(z)) \;\geq\; \sum_{z=1}^{x-1} (d(z) - m).$$

Hence

$$\text{first}(x) \geq \frac{1}{2m} \sum_{z=1}^{x-1} d(z).$$

Let C be some generic constant whose value can vary at different places of the proof. For Theorem 7.2 choosing $\epsilon' = \epsilon$, we have for all $x \geq 1$,

$$\sum_{z=1}^{x-1} d(z) \geq \sum_{z=1}^{x-1} \left(\frac{t}{z}\right)^{\eta(1-\epsilon')} = C t^{\eta(1-\epsilon')} x^{1-\eta(1-\epsilon')} = C x \left(\frac{t}{x}\right)^{\eta(1-\epsilon')}.$$

For Theorem 7.3 (web-graph case), choosing $\epsilon' = 0$ we have from Lemma 7.2 that for all $x \geq \log t$,

$$\sum_{z=1}^{x-1} d(z) \geq mx \left(\frac{t}{x}\right)^{\eta(1-\epsilon')}.$$

Thus in both cases, for all $x \geq \log t$,

$$\bar{d}(\text{first}(x)) = \left(\frac{t}{\text{first}(x)}\right)^{\eta} \leq C \left(\frac{t}{x}\right)^{\eta(1-\eta(1-\epsilon'))}. \tag{7.11}$$

Using (7.10) and (7.11), and the bound $\bar{d}(\text{first}(x)) = \widetilde{O}(t^{\eta})$ for $x \leq \log t$, we get

$$\begin{aligned} \bar{w}(G^*) &= O(\text{polylog}(t)) \cdot \sum_{x=1}^{t} \left(\frac{t}{x}\right)^{\eta(1+b)} \left(\frac{t}{x}\right)^{b\eta(1-\eta(1-\epsilon'))} \\ &= O(\text{polylog}(t)) \cdot \sum_{x=1}^{t} \left(\frac{t}{x}\right)^{\eta(1+b(2-\eta(1-\epsilon')))}. \end{aligned} \tag{7.12}$$

Choosing

$$\eta(1 + b(2 - \eta(1 - \epsilon'))) = 1, \tag{7.13}$$

the sum in (7.12) is $O(t \log t)$. This, (7.8) and (7.9) imply $w(G) = \widetilde{O}(t)$.

□

Theorem 7.2. *Let $G(t)$ be a pseudo-preferential graph with degree sequence satisfying (7.1). Let $S_a = \{v : d(v) \geq t^a\}$ be connected with diameter $Diam(S_a)$. Let $b = (1 - \eta)/(\eta(2 - \eta(1 - \epsilon)))$.*

*A biased seeded random walk with transition probability along edge $\{x, y\}$ proportional to $(d(x)d(y))^b$, finds all vertices in $G(t)$ of degree at least t^a in $\widetilde{O}(Diam(S_a) \times t^{1-2ab})$ steps, **whp**. The cover time of the graph $G(t)$ by this biased walk is $\widetilde{O}(tDiam(G(t)))$.*

Theorem 7.3. *Let $c \geq 3$, $m \geq 2$, $a < 1/(c-1)$ and $\epsilon = (1 + 1/a - c)/m$. Let $b = (c-1)(c-2)/(2c-3)$.*

*A biased seeded random walk with transition probability along edge $\{x, y\}$ proportional to $(d(x)d(y))^b$, finds all vertices in $G(c, m, t)$ of degree at least t^a in $\widetilde{O}(t^{1-2ab(1-\epsilon)})$ steps, **whp**. The cover time of the graph $G(c, m, t)$ by this biased walk is $\widetilde{O}(t)$.*

Proof

We apply the Matthews bound (3.30) to set S_a. Clearly $\log |S_a| \leq \log t$. It remains to find

$$\max_{u,v \in S_a} H(u, v) \leq \max_{u,v \in S_a} K(u, v).$$

To calculate $K(u, v)$ in (3.29), we use the bound of $w(G)$ from Theorem 7.1.

The set S_a is connected with diameter $\mathrm{Diam}(S_a)$ as specified. Let $\Delta(a) = \mathrm{Diam}(S_a)$, then for any $u, v \in S_a$ there is a path uPv of length $O(\Delta(a))$ from u to v in $G(t)$ contained in S_a, and thus consisting of vertices w of degree $d(w, t) \geq t^a$. Thus each edge (x, y) of this path has resistance $1/(d(x)d(y))^b \leq 1/t^{2ab}$, so

$$R_{\mathit{eff}}(u, v) \leq \mathrm{Diam}(S_a) \cdot t^{-2ab}. \tag{7.14}$$

The Matthews bound (3.30), Theorem 7.1 and the bound (7.14) give the (expected) seeded cover time $C^*_{S_a} = \widetilde{O}(\mathrm{Diam}(S_a) \cdot t^{1-2ab})$.

Suppose we want to find all vertices of degree at least t^a for some $a > 0$ in $G(t) \equiv G(c, m, t)$. Recall that $G(t)$ is generated by a process of attaching v_t to $G(t-1)$. At what steps were the vertices $v \in S_a$ added to $G(t)$? The expected degree of v at step t is given by (2.17) i.e. $\mathbf{E}d(v, t) = (1 + o(1))m(t/v)^\eta$. This function is monotone decreasing with increasing v. Let σ be given by

$$t^a = \left(\frac{t}{\sigma}\right)^\eta \qquad \text{or equivalently} \qquad \sigma = t^{1-a/\eta}. \tag{7.15}$$

Let $s = \sigma \cdot \log^{2/\eta+1} t$, then using Lemma 7.1 all vertices added at steps $w \geq s$ have $d(w,t) = o(t^a)$, **whp**. On the other hand, using Lemma 7.1 again, all vertices v added at steps $1, ..., s$ have degree $d(v,t) \geq (t/s)^{\eta(1-\epsilon)}$.

For Theorem 7.3 let $\Delta(a) = \text{Diam}(G(s))$ where s is defined above. Because $\text{Diam}(G(s)) = O(\log s)$, (see (2.19)), we know that for any $u, v \in S_a$ there is a path uPv of length $O(\log t)$ from u to v in $G(t)$ contained in $G(s)$, and thus consisting of vertices w of degree $d(w,t) \geq (t/s)^{\eta(1-\epsilon)} = t^{a(1-\epsilon)}/\text{polylog}(t)$. Thus each edge (x,y) of this path has resistance $1/(d(x)d(y))^b \leq \text{polylog}(t)/t^{2ab(1-\epsilon)}$, so

$$R_{\mathit{eff}}(u,v) \leq t^{-2ab(1-\epsilon)} \cdot \text{polylog}(t). \tag{7.16}$$

The Matthews bound (3.30), Theorem 7.1 and the bound (7.16) give the (expected) seeded cover time $C^*_{S_a} = \widetilde{O}(t^{1-2ab(1-\epsilon)})$. In both cases, apply the Markov inequality ($\mathbf{Pr}(X > A \cdot \mathbf{E}X) \leq 1/A$), with $\mathbf{E}X = C^*_{S_a}$, and $A = \log t$ to give a **whp** result, that all vertices of degree at least t^a can be found in time

$$T_{PP}(a) = \widetilde{O}(\text{Diam}(S_a) \cdot t^{1-2ab}),$$

in pseudo-preferential graphs, and in time

$$T_{WG}(a) = \widetilde{O}(t^{1-2ba(1-\epsilon)}),$$

in web-graphs.

For preferential attachment graphs $\eta = 1/2$, and (7.13) gives $b = 2/3$, and the time $T_{PA}(a)$ is

$$T_{PA} = \widetilde{O}(t^{1-(4/3)a(1-\epsilon)}).$$

Finally we establish the cover time of the graph $G(t)$. This is done by using (3.30) with $S = V(t)$ the vertex set of $G(t)$, i.e.

$$C_{V(t)} \leq \max_{u,v \in V(t)} H(u,v) \log t. \tag{7.17}$$

We bound $H(u,v)$ by (3.29) as usual. The resistance $r(e)$ of any edge $e = \{x,y\}$ is

$$r(e) = \frac{1}{(d(x)d(y))^b} \leq \frac{1}{m^{2b}} = O(1).$$

Let the diameter of $G(t)$ be $\mathrm{Diam}(G)$ which is specified for Theorem 7.2 and is $O(\log t)$ (**whp**) for Theorem 7.3. Thus $R_{\mathit{eff}}(u,v) = O(\mathrm{Diam}(G))$, since the effective resistance between u and v is at most the resistance of a shortest path between u and v. This and the bound $w(G) = \widetilde{O}(t)$ proven in Theorem 7.1 give $K(u,v) = \widetilde{O}(t \cdot \mathrm{Diam}(G))$. Thus the cover time of the graph $G(t)$ is $\widetilde{O}(t \cdot \mathrm{Diam}(G))$.

□

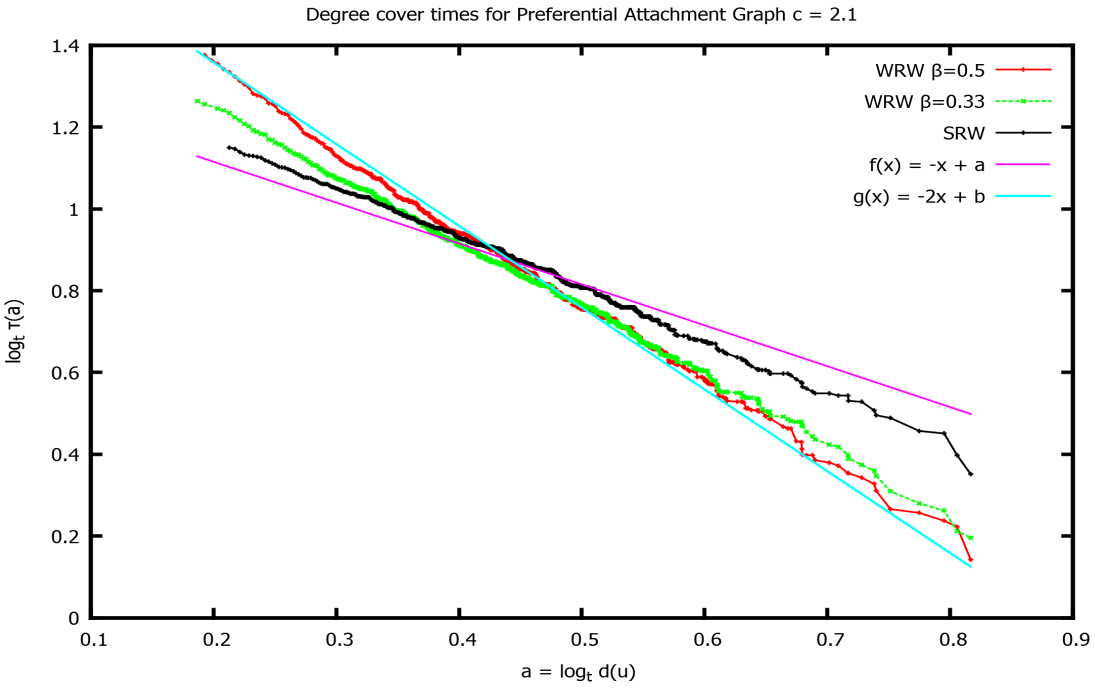
7.1.3 Experimental results

Theorem 7.3 gives an encouraging upper bound of the order of around $t^{1-(4/3)a}$ for a biased random walk to cover all vertices of degree at least t^a in the t-vertex preferential attachment graph $G(c,m,t)$. Our experiments, summarized in Figures 7.2-7.5, suggest that the actual bound is stronger than this. The experiments were made on various preferential attachment graphs $G(c,m,t)$ with $m = M = 3$, and $t = 6 \times 10^5$ vertices. There were various values of c The degree distribution of these graphs was presented Chapter 2, Section 2.2.2 in Page 48 and follows Equation (2.1), repeated below:

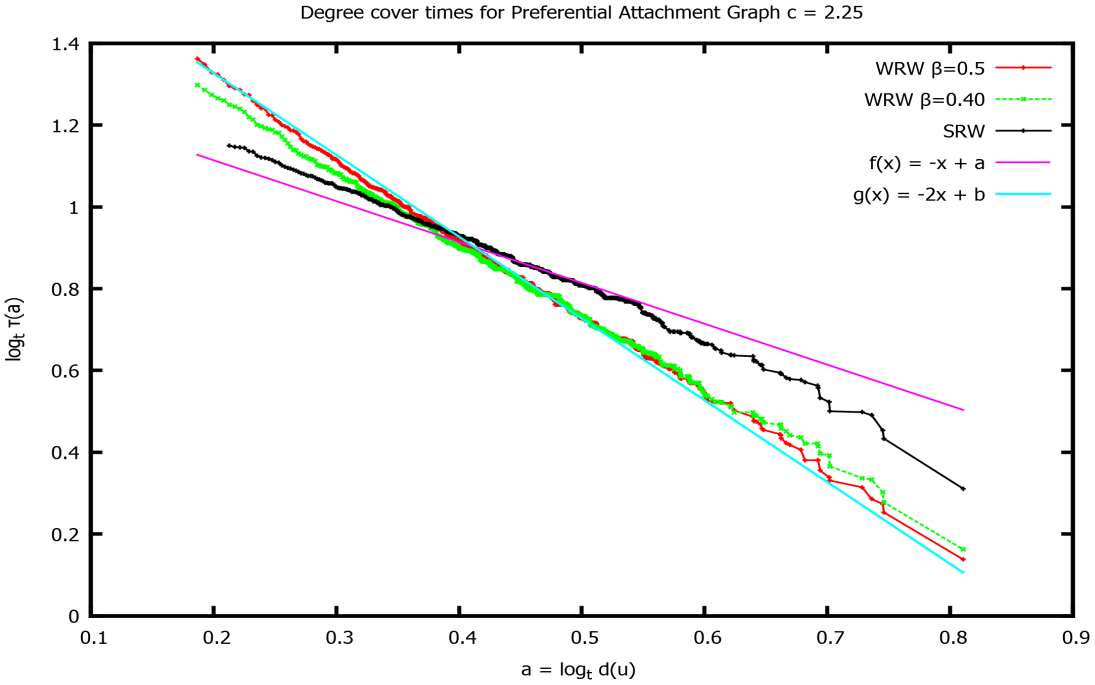
$$n_m(\ell) = \frac{\Gamma(\xi + \frac{1}{\eta})}{\eta\Gamma(\xi)} \frac{\Gamma(\ell+\xi)}{\Gamma(\ell+\xi+1+\frac{1}{\eta})}$$

where $\xi = \xi(u) = m(u) + \frac{\nu}{\eta}$ and $m = m(u)$ is the degree of vertex u when first added.

We remind that ν is the proportion of edge endpoints added to the graph *uar* while η is the proportion added *preferentially*. We have used values of η ranging from 0.1 to 0.9. For each of these values, we used three different random walks. A SRW, a WRW with $\beta = \frac{1}{2}$ and a WRW with $\beta = \beta_{opt} = \frac{1\eta}{\eta(2\eta)}$. We see the results of these experiments in Figures 7.2-7.5. Both axes are in logarithmic scale. The y-axis is $y = (\log \tau(a))/\log t$. There are also two reference lines drawn in each of the Figures in 7.2-7.5. These lines have slopes $-a$ and $-2a$, and are included for visual inspection only. To calculate the speed up, given $x = a$, read off the $y(a)$-values y_S, y_W. The speed up is $t^{y_S - y_W}$, where $t = 6 \times 10^5$. Curiously, the improvement does not seem sensitive to the precise value of β_{opt} and the difference between $\beta = \frac{1}{2}$ and $\beta = \beta_{opt}$ is only apparent if $\beta_{opt} \gg \frac{1}{2}$.

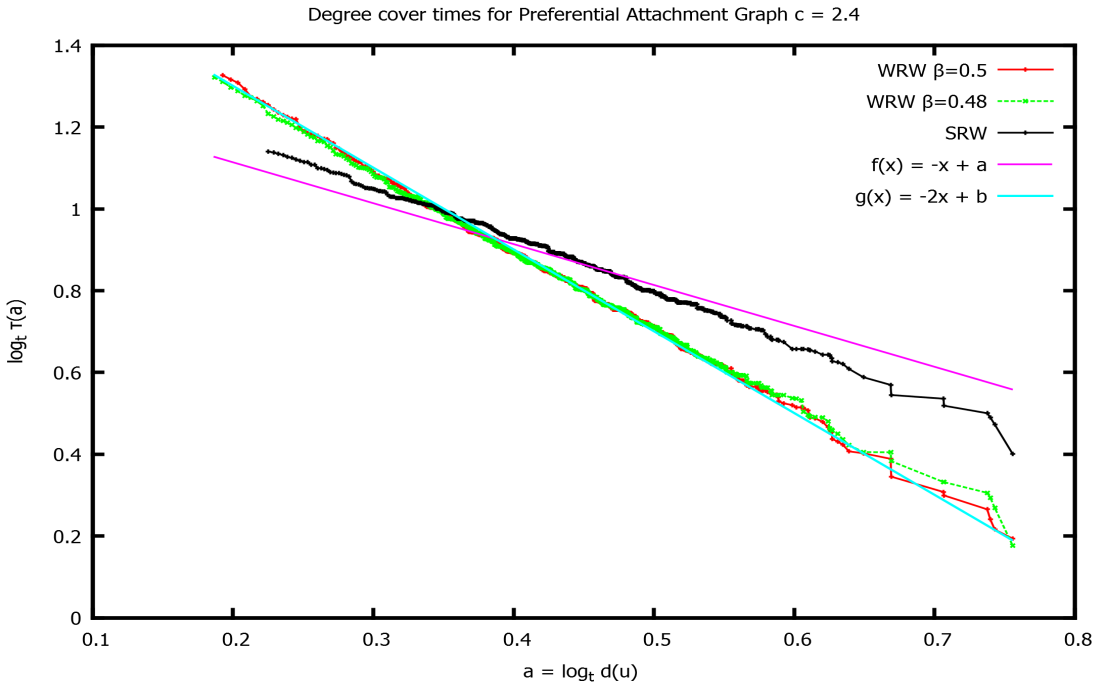


(a) $\eta = 0.9$ ($c = 2.1$)

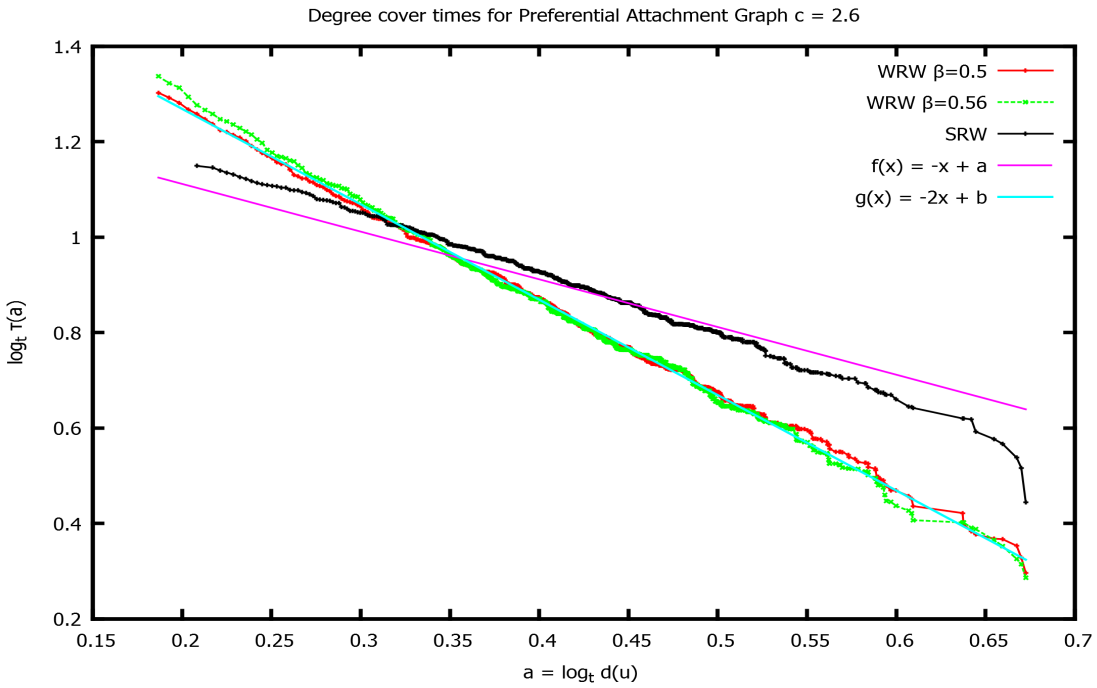


(b) $\eta = 0.8$ ($c = 2.25$)

Figure 7.2: Cover time of all vertices of degree at least t^a in $G(c, m, t)$, as a function of a.

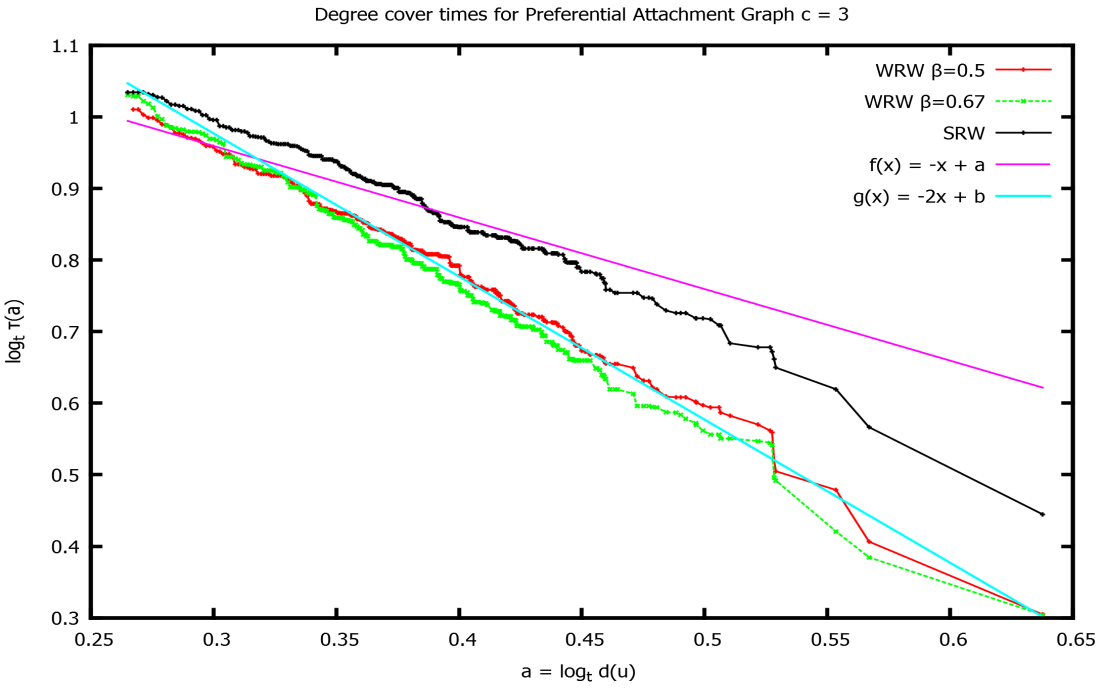


(a) $\eta = 0.7$ ($c = 2.4$)

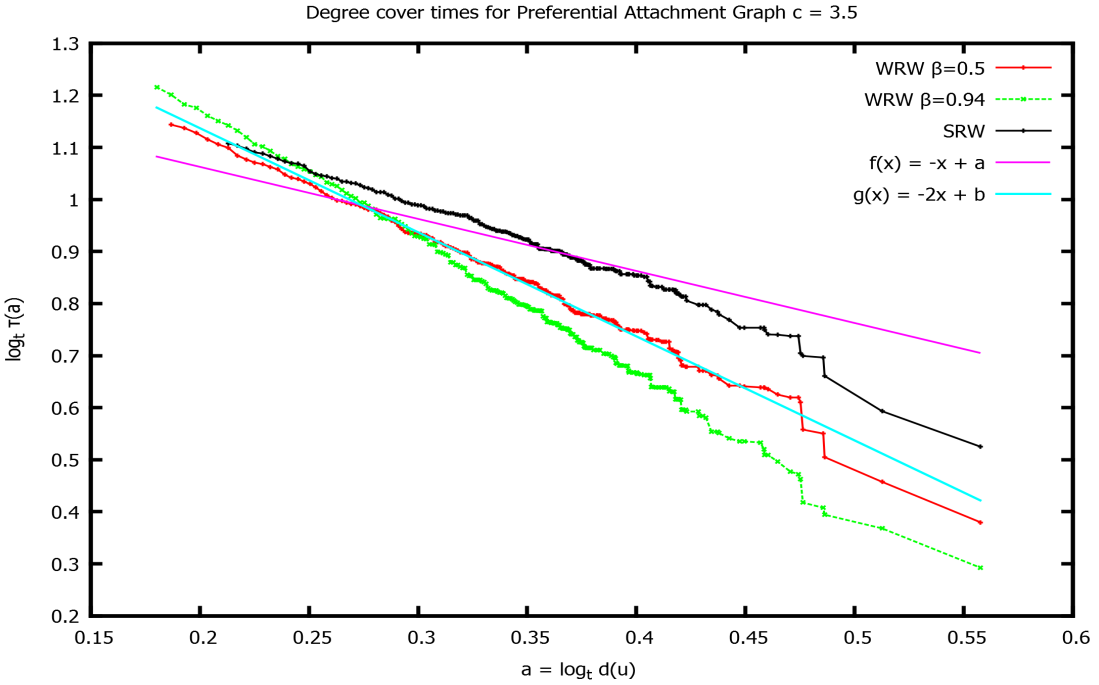


(b) $\eta = 0.6$ ($c = 2.6$)

Figure 7.3: Cover time of all vertices of degree at least t^a in $G(c, m, t)$, as a function of a.

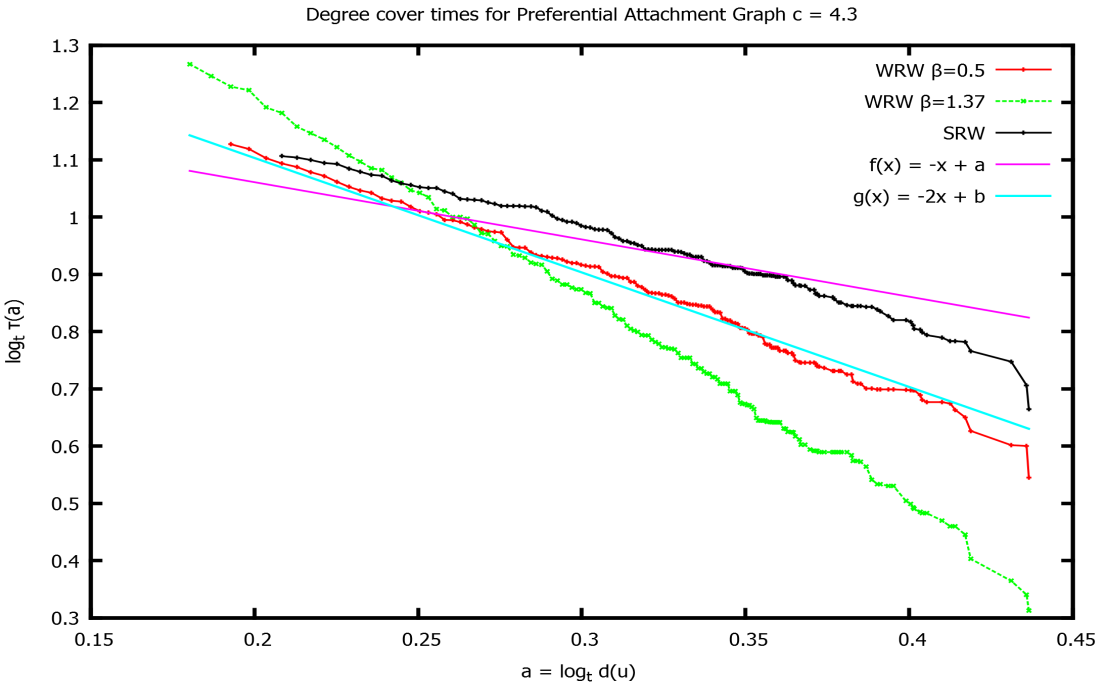


(a) $\eta = 0.5$ $(c = 3)$

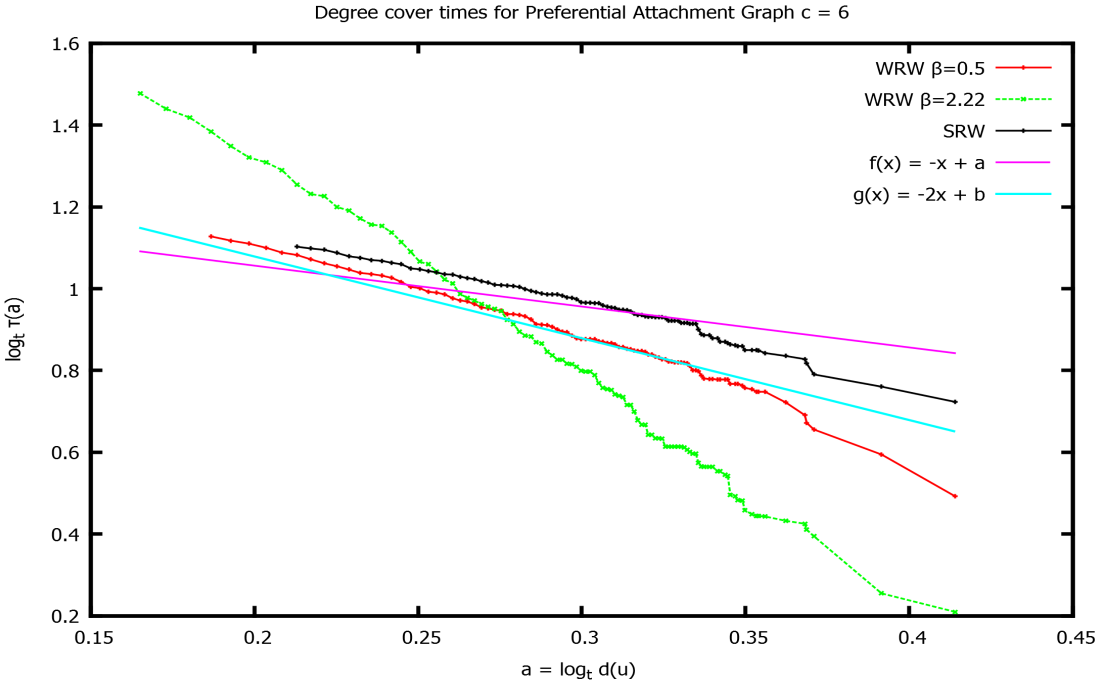


(b) $\eta = 0.4$ $(c = 3.5)$

Figure 7.4: Cover time of all vertices of degree at least t^a in $G(c, m, t)$, as a function of a.



(a) $\eta = 0.3$ ($c = 4.3$)



(b) $\eta = 0.2$ ($c = 6$)

Figure 7.5: Cover time of all vertices of degree at least t^a in $G(c, m, t)$, as a function of a.

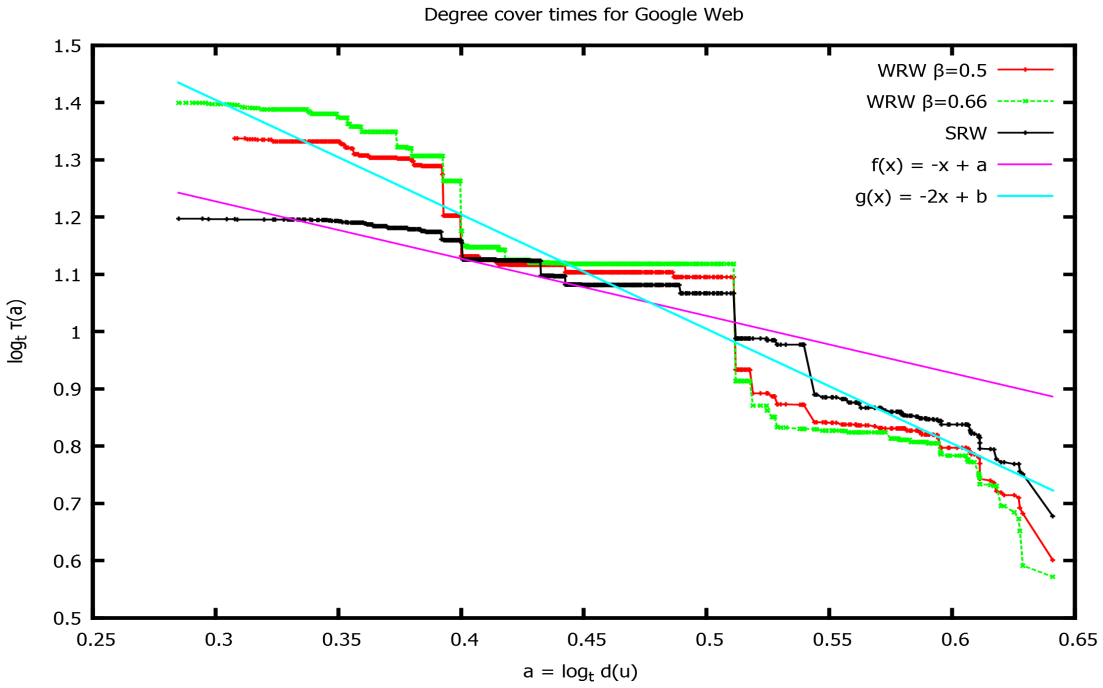


Figure 7.6: Google Web Sample: Cover time of all vertices of degree at least t^a in G_W as a function of a

Our experimental results for Theorem 7.2 are less clear cut, but still encouraging. Figure 7.1a gives the degree distribution of the underlying graph of the WWW, on $t = 8.7 \times 10^5$ vertices obtained from the SNAP[1] database. The power law exponent is approximately $c = 3$, and it was crawled using a weight of $b = 2/3$. Figure 7.6, shows the results obtained by averaging 25 runs of the simple and weighted random walks. The weighted walk is generally about 4 times faster for $a > 0.43$.

Before discussing Figures 7.2-7.5 in greater detail, we remark that they broadly confirm the implications from our theoretical analysis: for random preferential attachment graphs, biased random walks discover quickly all higher degree vertices while not increasing by much the cover time of the whole graph. For example, by checking the exact cover times, we observed that the biased random walk with $b = 1/2$ took on average 2.7 times longer than a simple random walk to cover the whole graph $G(3, 3, 6 * 10^5)$, but

[1] `http://snap.stanford.edu/data/web-Google.html`

discovered the 100 highest degree vertices 10 times faster than a simple random walk.

For a weighted random walk, the stationary distribution $\pi(v)$ of vertex v is given by

$$\pi(v) = \frac{1}{w(G)} \sum_{x \in N(v)} w(v,x),$$

where $w(G)$ is the sum of the edge weights of G, each edge counted twice. Thus for a simple random walk on $G(m,t)$, $\pi_S(v) = d(v)/2mt$. For the weighted random walk of Theorem 7.2 ($\eta = 1/2$ and $b = 1/2$ for $G(m,t)$) we have the following lower bound.

$$\pi_W(v) = \Omega((d(v))^{3/2}/(t \operatorname{polylog}(t))).$$

This bound holds because we know from Theorem 7.1 that $w(G) = \widetilde{O}(t)$, and

$$\begin{aligned} \sum_{x \in N(v)} w(v,x) &= \sum_{x \in N(v)} (d(v)d(x))^{1/2} \\ &\geq (d(v))^{1/2} \sum_{x \in N(v)} (m)^{1/2} \\ &\geq (d(v))^{3/2}. \end{aligned}$$

We can give an informal explanation of Figures 7.2-7.5 as follows. In the long run, the number of visits to vertex v in T steps approaches $T\pi(v)$, so the first visit to v should be at about $T(v) = 1/\pi(v)$. As $\pi(v)$ increases with increasing degree $d(v)$, then if $h > a$ we should expect to see all vertices of degree t^h before all vertices of degree t^a.

For a simple random walk, let v be a vertex of degree t^a, then $T(v) \approx 1/\pi_S(v) = 2mt/t^a \approx t^{1-a}$. So the SRW plot in Figure 7.2 should have slope $-a$, and this is indeed the case.

For a weighted random walk, the same argument gives

$$\frac{1}{\pi_W(v)} = O\left(\frac{t \log^5 t}{(t^a)^{3/2}}\right) = \widetilde{O}(t^{1-3/2a}),$$

which explains the slope of about $-3a/2$ for (at least part of) the WRW plot.

The total number $n(a)$ of vertices of degree at least t^a is approximated by $\sigma = t^{1-a/\eta} = t^{1-2a}$, where the value of σ from (7.15), is the expected step at which a vertex of degree

t^a is added, and $\eta = 1/2$ for preferential attachment. As no walk based process can visit σ vertices in less than σ steps, this explains our inclusion of the line with slope $-2a$ in Figures 7.2–7.5.

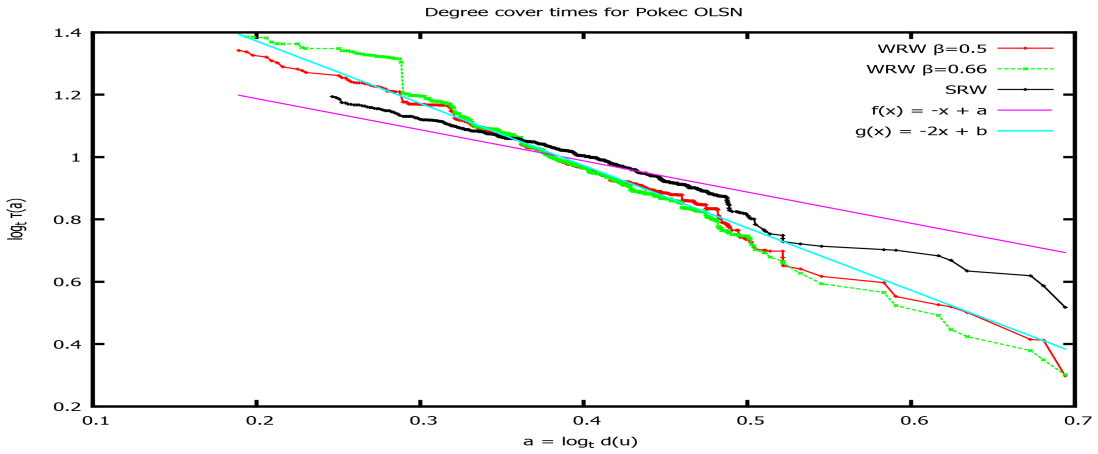
7.2 Finding high degree vertices in general graphs with a power-law degree distribution

In this section we give a discussion to support a general heuristic which chooses a value of $b = 1/2$ for a weighted random walk to find high degree vertices in heavy tailed graphs with power law $c \geq 3$.

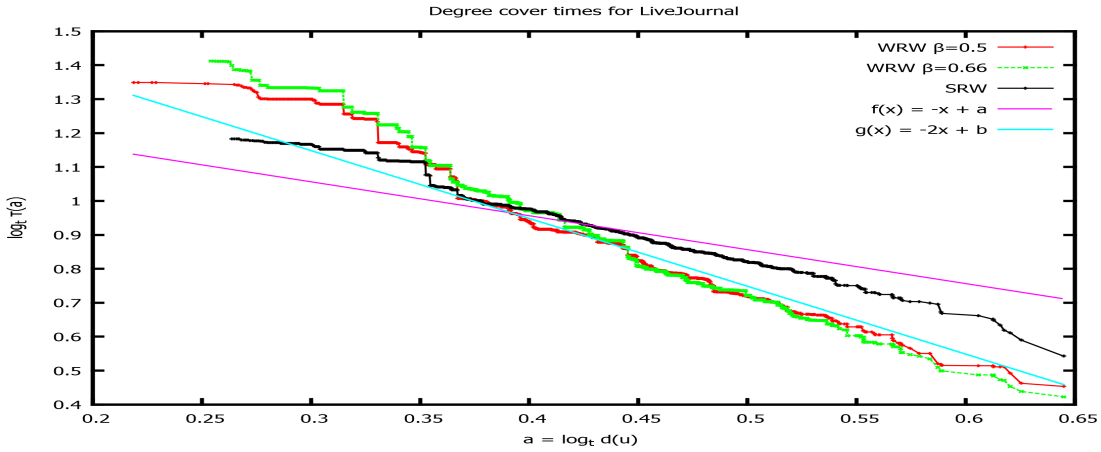
The setting is a graph G for which the general properties (number of edges, vertices, degree sequence etc.) are unknown, but which we suppose to be a 'small world network', in the sense that it is connected, with power law degree sequence and the number of edges is linear in the number of vertices. We suppose as before, that we want to find all high degree vertices of G as quickly as possible. Theorems 7.2, 7.3 do not provide any insight into the best value of b to choose, even if, as in Theorem 7.3, we assume that the power law parameter $c \geq 3$. For $c \geq 3$ at least, there is theoretical evidence to support the choice of $b = 1/2$, as we now explain.

The basic assumptions we make are:
(i) We regard the power law parameter c, the maximum degree Δ and the number of vertices n as unknowns which cannot be used as inputs into the choice of b.
(ii) We assume G has a power law parameter c and heavy tailed degree sequence $(n_1, n_2, ..., n_\Delta)$, where $n_k \propto nk^{-c}$.
(iii) We assume the number of edges $m = an$ for some constant $a > 1$.

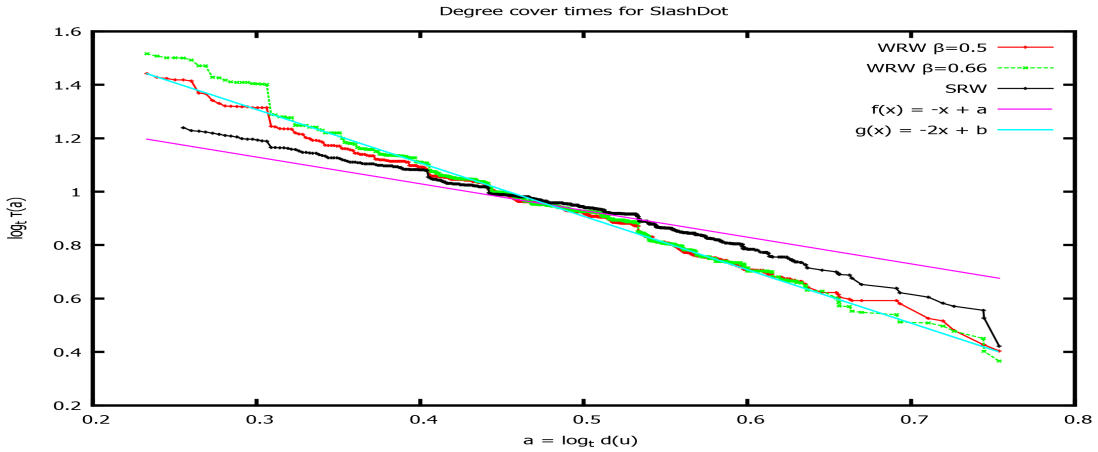
We remark that, for graphs with a power-law degree sequence, the value of Δ needs to be determined independently of c. For example, in many cases on power-law degree sequences an arbitrary assumption is made that $\Delta = \widetilde{O}(n^{1/2})$ further details on the upper bound of Δ can be found in the work of A. Flaxman *et al.* [48]. In the web-graph model of [24] the **whp** value of $\Delta = \widetilde{O}(n^{1/(c-1)})$ is a property of the model.



(a) Pokec OLSN



(b) LiveJournal OLSN



(c) SlashDot OLSN

Figure 7.7: Cover time of all vertices of degree at least t^a in G_W as a function of a

An upper bound on the time taken by a reversible random walk to hit a vertex v, from arbitrary starting position can be obtained from the following lemma. See e.g. [27] for a proof of this lemma.

Lemma 7.3. *Let W be a lazy random walk with second eigenvalue $\lambda = \lambda(W)$ and stationary distribution π. With high probability, all vertices v with stationary distribution $\pi_v \geq \rho$ can be found in $T(\rho)$ steps, where*

$$T(\rho) = O\left(\frac{\log n}{\rho\,(1-\lambda)}\right). \tag{7.18}$$

For a weighted random walk W, the stationary distribution of vertex v is given by

$$\pi_v(W) = \frac{\sum_{u\sim v} w(u,v)}{\sum_{x\in V}\sum_{y\sim x} w(x,y)} = \frac{w_v}{w_G},$$

where $x \sim y$ means the edge (x,y) exists.

In order to bias the walk towards high degree vertices we consider weights $(d(x)d(y))^b$ for edges (x,y), where $b > 0$. From the Cauchy-Schwarz inequality,

$$w_G = \sum_{x\in V}\sum_{y\sim x}(d(x)d(y))^b \leq \sum_{x\in V}(d(x))^{2b+1},$$

and, as $d(u) \geq 1$,

$$w_v = \sum_{u\sim v}(d(u)d(v))^b \geq (d(v))^{b+1}.$$

Thus

$$\pi_v(W) = \frac{w_v}{w_G} \geq \frac{(d(v))^{b+1}}{\sum_{x\in V}(d(x))^{2b+1}}. \tag{7.19}$$

The equivalent simple random walk, S, has stationary distribution $\pi_v(S) = d(v)/2m$. For a weighted walk to reach high degree vertices faster than a simple random walk, we would need to choose a value b for the edge weights in such a way that,

$$\pi_v(W) \gg \pi_v(S) = \frac{d(v)}{2m}, \tag{7.20}$$

If we do this, and provided $\lambda(W)$ is not too different from $\lambda(S)$, then by Lemma 7.3, the weighted walk improves the hitting time of such vertices in comparison to a simple random walk.

Let $\phi(b) = \sum_{x \in V} (d(x))^{2b+1}$. Using (7.19) and the assumption that $m = an,\ a > 1$ we can make a worst case analysis, replacing the requirement (7.20) by

$$\pi_v(W) = \frac{w_v}{w_G} \geq \frac{(d(v))^{b+1}}{\phi(b)} \gg \frac{d(v)}{2m} = \frac{d(v)}{2an},$$

which is equivalent to

$$(d(v))^b \gg \frac{\phi(b)}{n}. \tag{7.21}$$

Under the power-law degree sequence assumption, the sum $\phi(b)$ in the denominator of (7.19) can be approximated in the following way. We approximate the number of vertices of degree k by $N_k \sim nAk^{-c}$ (see Equation (2.1) in Page 35). Therefore:

$$\begin{aligned}
\phi(b) &= \sum_{x \in V} (d(x))^{2b+1} \\
&= \sum_{k=1}^{\Delta} k^{2b+1} N_k \\
&\approx nA \int_1^{\Delta} k^{-c} k^{2b+1} \, \mathrm{d}k \\
&= nA \int_1^{\Delta} k^{2b+1-c} \, \mathrm{d}k
\end{aligned}$$

In the case that $c = 2b + 2$:

$$\begin{aligned}
\phi(b) &= nA \int_1^{\Delta} k^{2b+1-(2b+2)} \, \mathrm{d}k \\
&= nA \int_1^{\Delta} k^{-1} \, \mathrm{d}k \\
&= nA \left(\log k |_1^{\Delta} \right) \\
&= nA \log \Delta \approx \frac{1}{2} nA \log n
\end{aligned}$$

since $\Delta \sim n^{\frac{1}{2}}$.

In all other cases:

$$\begin{aligned}
\phi(b) &= \frac{nA}{2b+2-c} \left(-k^{2b+2-c} \Big|_1^{\Delta} \right) \\
&= \frac{nA}{2b+2-c} (\Delta^{2b+2-c} - 1)
\end{aligned}$$

If $c > 2b+2$ then $\Delta^{2b+2-c} \ll 1$. In the case that $c < 2b+2$ then $\Delta^{2b+2-c} \gg 1$. Therefore $\phi(b)$ can be approximated by

$$\phi(b) \propto \begin{cases} n & c > 2b+2 \\ n \log n & c = 2b+2 \\ n\Delta^{2b+2-c} & c < 2b+2 \end{cases}.$$

Thus we can rewrite (7.21) as

$$(d(v))^b = \Omega(\log n \ \max(1, \Delta^{2b+2-c})).$$

At this point it becomes clear that there is no perfect answer. For given c, we need to choose values of b satisfying $c \geq 2b+2$.

The results of Theorem 7.3 apply when $c \geq 3$. To generalize these results for heuristic use, we assume $c \geq 3$. The value $b = 1/2$ satisfies our worst case analysis, and seems the best choice for that range. We suggest its use as an informed heuristic.

Interestingly, the value $b = 1/2$ works quite well experimentally, not only on graphs with parameter $c = 3$ (see Figures 7.2–7.5, 7.6), but also marginally improves on a simple random walk graphs with much lower values of c such as a SNAP[2] copy of the SlashDot, LiveJournal and Pokec OLSNs networks (see Figure 7.7).

Perhaps the moral is, that any positive weight is good, as it biasses the walk towards high degree vertices.

When we change from a simple random walk S to a weighted walk W, we change the second eigenvalue $\lambda(W) = \lambda_2$ of the transition matrix, and hence the mixing rate. Although small world networks are regarded as having small diameter, which means they must be expanders, and rapidly mixing for simple random walks, it is difficult to estimate the change in mixing rate for general power law graphs.

Some bounds on the change in eigenvalue gap can be obtained from the *Direct Comparison Lemma* for weighted random walks (Lemma 29, Ch 3 [2]). This lemma was

[2] `http://snap.stanford.edu/data/index.html`. Data retrieved 15/12/2011

introduced in Chapter 5 as Lemma 5.1.

$$\frac{\min_e(w_e/w_e^*)}{\max_v(w_v/w_v^*)} \leq \frac{\tau_2}{\tau_2^*} \leq \frac{\max_v(w_v/w_v^*)}{\min_e(w_e/w_e^*)}.$$

where τ_2 is the relaxation time of a random walk, given by $\tau_2 = 1/(1-\lambda_2)$.

The bounds on the change in mixing rate for the weighted walk we obtain in this way are a function of Δ. As we do not know Δ, it is difficult to quantify further.

7.2.1 Conclusions

Our theoretical analysis of the number of steps required by biased random walks to discover all high degree vertices in random t-vertex preferential attachment graphs, was restricted to power laws $c \geq 3$. This was a consequence of the proof technique. The statement of Theorem 7.3 can be used to give a value of b for the walk weight for any $c > 2$, but we do not have a proof of correctness for $c < 3$. A method which provably worked down to $c = 2$, or even lower would be desirable; especially as power laws around $c = 2$ are common in real networks [16].

As a heuristic, the method we propose seems to work well with weight $b = 1/2$ in many cases where there is no performance guarantee. An example of this is the SlashDot network which has a power law of $c = 1.6$. At a guess, the performance of our method depends more on the existence of a dense subgraph between high degree vertices, than on the exact power law of the degree sequence. The existence of such a dense subgraph is important to the discussion in Section 7.1.2 as it means the effective resistance between high degree vertices is small, thus ensuring a low cover time in (3.30).

There is a lack of agreement among models as to the range of values for high degree vertices for a given power law c. To give an example, the web-graph model of [24] predicts vertices of degree $\Theta(t^{1/(c-1)})$ on t-vertex graphs, but some authors arbitrarily truncate heavy tailed degree sequences at $O(t^{1/2})$. Any technique to find such vertices would have to be tuned to exploit the related structures in the graph.

Chapter 8

Information filtering and recommendation

A recommender system uses information about known associations between *users* and *items* to compute for a given user an ordered recommendation list of items which this user might be interested in acquiring. For example, in the context of a library, knowing which books have been read by which members of the library, a recommender system would calculate for a given member an ordered list of book recommendations for next reading. The act of using known information as a means to predict a future state of a system is what is commonly known as an information prediction system or a recommender system.

The ideas developed are joint work with C. Cooper, S. H. Lee and T. Radzik and appear in [34]. This work has been funded by Samsung as part of the Samsung Global Research Outreach programme awarded in 2012 for the project "Fast low cost methods to learn structure of large networks".

8.1 Recommendation Systems Based On Collaborative Filtering

We view a recommender system as an algorithm which takes a dataset of relationships between a set of *users* and a set of *items* and attempts to calculate how a given user might rank all items. For example, the users may be the customers who have bought books

from some (online) bookstore and the items the books offered. The core information in the dataset in this case would show who bought which books, but it may also include further details of transactions (the date of transaction, the books bought together, etc.), information about the books (authors, category, etc.), and possibly some details about customers (age, address, etc.). For a given customer, a recommender system would compute a list of books $\langle m_1, m_2, \ldots, m_k \rangle$ which this customer might be interested in buying, giving the highest recommendations first. Recommender systems viewed as algorithms for computing such personalised rankings of items (rather than overall "systems," which would also include methods for gathering data) are often referred to as *scoring*, or *ranking algorithms*.

Recommender systems are part of everyday online life. Whenever we buy a movie, or a new app for our mobile phone, a recommender system would suggest other items of potential interest to us. A good recommender system improves the user's experience and increases commercial activity, while consistently unhelpful recommendations may make the users look for other sites. This significant commercial value coupled with the challenging theoretical and practical aspects of modelling, designing and implementing appropriate algorithms, has made recommender systems a fast growing research topic.

We focus on a simple scenario with two main entity sets, Users (U) and Items (I), and a single relationship R of pairs $\langle u, m \rangle$, where $u \in U$ and $m \in I$. The fact that $\langle u, m \rangle \in R$ means that u has some preference for m. The pairs $\langle u, m \rangle \in R$ may have additional attributes which indicate the degree of preference. The relationship R can be modelled as a bipartite graph $G = (U \cup I, R)$, possibly with edge weights, which would be calculated on the basis of the attributes of pairs $\langle u, m \rangle \in R$. A scoring algorithm for a user $u \in U$ orders the items in I according to some similarities between vertex u and the vertices in I, which are defined by the structure of graph G. More precisely, a scoring algorithm is defined by a formula or a procedure for calculating an $n \times n$ matrix $M = M(G)$ which expresses those similarities, where $n = |U| + |I|$. For a user $u \in U$, the items $m \in I$ are ranked according to their $M(u, m)$ values (in increasing or decreasing order, depending whether a lower or a higher value $M(u, m)$ indicates higher or lower similarity between

vertices u and m). Matrix M is called the *scoring matrix* or the *ranking matrix* of the algorithm. We also note that the methodology of designing ranking algorithms of this type, which use the whole relationship between users and items rather than user and item profiles, is often referred to as *collaborative filtering*.

Let $\mathcal{W}_u = \langle X_0, X_1, \ldots, X_t, \ldots \rangle$ be a RW on graph G starting from vertex u. That is, $X_0 = u$ and X_{t+1} is a neighbour of X_t selected according to the RW transition probability. Let $h(u, v)$ denote the first step $t \geq 1$ such that $X_t = v$. The *hitting time* of v from u is the expectation $H(u, v)$ of the random variable $h(u, v)$. The *Laplacian matrix* of graph G is defined as

$$L = D - A$$

where A is the adjacency matrix of G and D is the diagonal matrix of the vertex degrees. The main scoring algorithms considered in Fouss *et al.* [50, 51] are the hitting-time algorithm, the reverse hitting-time algorithm, the commute-time algorithm and the L-pseudoinverse algorithm. The scoring matrices M of these algorithms are matrices H, H^T, $C = H + H^T$ and L^+, respectively. Matrix L^+ is the *Moore-Penrose pseudoinverse* [109, 115] of L, in this paper referred to as simply the *inverse Laplacian*. Detailed explanation of these ranking matrices can be found in Section 8.4. For all these matrices, a lower value of $M(u, m)$ indicates higher similarity between vertices u and m, and a higher rank of m in the ranking list of items for the user u. For example, the commute-time algorithm ranks the items for a user u according to the expected *commute times* $C(u, m) = H(u, m) + H(m, u) = H(u, m) + H^T(u, m)$, putting an item m_1 ahead of an item m_2, if $C(u, m_1) < C(u, m_2)$.

Fouss *et al.* [50, 51] reported that for the MovieLens dataset which they used in their experiments, the L^+ algorithm (and its variants) performed the best. The hitting-time and the commute-time algorithms performed similarly to the simple user-independent algorithm which orders the items according to their vertex degrees in the graph G. The reverse hitting-time algorithm showed the worst performance.

Kunegis and Schmidt [76] extend Fouss' work of [51] by taking user ratings into account

while computing a similarity measure on *users-items* bipartite graph. The similarity measure used in the paper is resistance distance, which is equivalent to the commute-time distance used in [51]. They adapt a rescaling method proposed in [9], in which the user ratings are rescaled into 1 (good) to -1 (bad). The authors conduct experiments on two datasets: 100K MovieLens dataset [111] and Jester [69] and the performance is measured by two evaluation metrics, the *mean squared error* and the *root mean squared error*.

Gori and Pucci [58, 59] present a RW based scoring algorithm for the recommendation systems. In [59] the algorithm is applied to movie recommendation whilst in [58], the algorithm is used for recommending research papers. For the movie recommendation, 100K (small) MovieLens dataset is used for the experiments and for the research paper recommendation, they use the dataset that is derived from the crawling of ACM portal website. The algorithm is based on the PageRank algorithm applied to an item similarity graph called a correlation graph. The correlation graph is constructed in [59] from the *users-items* bipartite graph, and in [58] from *citation* graph. Zhang *et al.* [133, 134] extend Gori and Pucci's work [59] by taking into account user's preference on item categories. The proposed algorithm computes the ranking scores based on the item genre and user interest profile.

Singh *et al.* [121] present an approach of combining a relations graph (friendship graph) with the ownership data (user-item graphs) to make recommendations. The combined graph is represented as an augmented bipartite graph and is treated as a Markov Chain with an absorbing state. The items are ranked according to the approximated absorbing distribution. This method is tested and evaluated for on-line gaming recommendation.

Lee *et al.* [79] consider a multidimensional recommendation problem, which allows some additional contextual information as an input. They present a RW based method, which adapts the Personalized PageRank algorithm. They conduct experiments on two datasets: *last.fm* [77] and *LG's OZ Store* [92]. The performance is evaluated using *hit at top-k* evaluation metric. Further work in this direction is reported in [80].

8.2 Dataset Description

Using the data collected by the MovieLens service [111], the GroupLens Research lab composed three datasets of user produced movie ratings [60]. These datasets are collections of quadruples $\langle$*UserId, MovieId, Rating, Timestamp*$\rangle$, which we view as pairs $\langle$*UserId, MovieId*$\rangle$ with attributes *Rating* and *Timestamp*. For each *UserId* and each *MovieId*, each dataset has at most one quadruple $\langle$*UserId, MovieId, Rating, Timestamp*$\rangle$. The three datasets have following sizes:

MovieLens 100K (basic set) – 943 users, $1,682$ movies, $100K$ ratings;

MovieLens 1M (medium size) – $6,040$ users, $3,900$ movies, $1M$ ratings;

MovieLens 10M (large size) – 71567 users, $10,681$ movies, $10M$ ratings.

The main use of these datasets is for the purposes of building and testing information prediction/recommendation systems as defined Section 8.4.

8.2.1 Ranking methodologies

To ensure reproducibility of results and to facilitate comparison between various recommender systems, each MovieLens dataset is partitioned into two parts: a training, or base, set B and a test set T. The test set is obtained by selecting 10 random user-movie ratings for each user. The training set consists of all remaining ratings. To make this partitioning feasible, the small dataset has more than 10 ratings per each user. The medium and large datasets have at least 20 ratings per each user.

The intended methodology for evaluating the performance of a recommender system is to consider the training set as the information about the past (which movies have been watched, and ranked, by each user) and the test set as the (hidden) information about who will watch what in the future. A recommender system is run on the training set to compute for each user a ranking of movies, and then the computed rankings are compared

with the test set. A better recommender system would be more effective in reconstructing the hidden information, that is, would give a closer fit between the computed rankings and the test set. More precisely, for each user u, the ranking of movies computed for this user is compared with the set $\{\overline{m}_1^{(u)}, \overline{m}_2^{(u)}, \ldots, \overline{m}_{t_u}^{(u)}\}$ of the test movies for this user. The pairs $\langle u, \overline{m}_1^{(u)}\rangle$, $\langle u, \overline{m}_2^{(u)}\rangle$, $\ldots$, $\langle u, \overline{m}_{t_u}^{(u)}\rangle$ are this user's ratings excluded from the training set and included in the test set. A better recommender system would place the movies $\overline{m}_i^{(u)}$ higher in the ranking. It is not obvious, however, how the closeness between the computed rankings and the test set should be quantified and a number of measures have been proposed.

Fouss *et al.* [50,51] evaluated the performance of ranking algorithms on the basic MovieLens dataset, taking into account all pairs $\langle \mathit{UserId}, \mathit{MovieId}\rangle$ included in this set, but ignoring the timestamp and rating attributes. The measure used in [50, 51] is the percentage of *correctly placed pairs*. In Section 8.2.1 we give the definition of this measure and the other measure which we use in this paper.

Herlocker *et al.* [65] discusses different measures of the performance of ranking algorithms. Among the most common ones are the percentage of *correctly placed pairs* (used in [50, 51]) and the number of *hits in the top k recommendations*. We use both these measures in this paper. To define these measures, we assume the general evaluation methodology described earlier (the bipartite graph $G = (U \cup I, R)$ partitioned into the training set B and the test set T), and we use the following notation: $|U| = p$, $|I| = q$, $|R| = r$, $I^{(u)} = \{m \in I : \langle u, m\rangle \in R\}$ is the set of all items associated with user u, $I_B^{(u)} = \{m \in I : \langle u, m\rangle \in B\}$ is the set of the training items for user u, $I_T^{(u)} = \{m \in I : \langle u, m\rangle \in T\}$ is the set of the test items for user u, $|I^{(u)}| = i_u$, $|I_B^{(u)}| = b_u$, $|I_T^{(u)}| = t_u$. For example, for a MovieLens dataset, $I^{(u)}$ is the set of all movies watched (rated) by user u, while $I_T^{(u)}$ is the set of the movies watched by user u but put aside in the test set. For all three MovieLens dataset, $t_u = 10$ for each user u.

For a given ranking algorithm $\mathcal{A}$, $\mathit{Rank}^{(u)} = \langle\, m_1^{(u)}, m_2^{(u)}, \ldots, m_{q_u}^{(u)} \,\rangle$ denotes the ranking of items computed for user u, and $m' <_u m''$ means that item m' appears in $\mathit{Rank}^{(u)}$

before item m'' (so is ranked higher than item m'').

8.2.2 Metric I: The number of correctly placed pairs

This measure requires that the ranking algorithm $\mathcal{A}$ computes for each user a full ranking, that is, a ranking which includes all items. Hence we assume $q_u = q$, for each user u. To calculate the score for a given user u, consider all pairs of items $\langle m', m'' \rangle$, with $m' \in I_T^{(u)}$ and $m'' \in I \setminus I^{(u)}$. We say that such pair $\langle m', m'' \rangle$ is *correctly placed*, if $m' <_u m''$, that is, if the (test) item m' is ranked higher than (appears before) the item m''. The score for user u is the percentage of correctly placed pairs:

$$\frac{|\{\langle m', m'' \rangle : \ m' \in I_T^{(u)}, \ m'' \in I \setminus I^{(u)}, \ m' <_u m''\}|}{t_u(q - i_u)} \cdot 100\%.$$

The score of the ranking algorithm is the average user score, over all user. If the t_u test items for user u are the first items, in any order, in the ranking $Rank^{(u)} \setminus I_B^{(u)}$, then the score for this user is the perfect 100%. For the MovieLens dataset, the score of 100% means that the top t_u movies recommended for user u, on the basis of the training set, are exactly the movies which this user has actually already watched, but have been hidden in the test set.

We note that if the items are ranked completely randomly (that is, the ranking is a random permutation of items), then for any pair of items is m' and m'', the probability that $m' <_u m''$ is equal to 1/2, so the (expected) score is 50%.

8.2.3 Metric II: The number of hits in the top k recommendations

An alternative evaluation measure is the number of *hits in the top k recommendations*. This measure is based on the simple all-or-nothing concept of counting only the test items which have made into the top k recommendations, ignoring their exact positions in the top k, and ignoring the positions of the test items outside the top k. The parameter k is either an absolute value, usually ranging between 10 and 100, or a fraction of the total number of items, usually ranging between 1% and 10%. We used both absolute

and relative values for k to compare the results that is the fraction or number of the items from $I_T^{(u)}$ which are among the first $k\%$ items or k items in the list $Rank^{(u)} \setminus I_B^{(u)}$.

If a random permutation of items is taken as the ranking, then the probability that a given test item is in the top k recommendations is equal to $k/(q - b_u)$. This is the expected score for user u, so the expected score of the random recommender is equal to $k \sum_u 1/(q - b_u)$, which is at least k/q and at most $k/(q - (r/p))$. The value r/p is the average degree of a user vertex in the graph G, so normally much less than q. Therefore, the expected score of the random recommender is approximately k/q.

8.2.4 Degree bias of T

The dataset provided comes pre-split into B and T. The method of doing such a split is also provided. Given the data-set D consisting of entries described in Section 8.2 the first 10 entries encountered for each user is placed in T and what remains becomes B. We note that while D is randomized, this specific method is degree biased since the probability of including a given movie in the test set is proportional to the probability that it's included in an randomly selected edge. Therefore this gives a convincing explanation on why arranging the movies according to their degree yields reasonably good results in [51].

To challenge the methods we have provided a different split to the graph by removing the 10 lowest degree movies from each user, while guaranteeing that we don't disconnect the graph. We note that in this case, ranking by degree performs no better than random ranking while at the same time the rankings provided by the other ranking methods discussed in [51] perform slightly better than random, while maintaining their relative order of evaluation score.

8.3 Summary of aims and findings

Our primary findings related to the above topics were as follows.

1. One particular measure which was good for [51] was the inverse Laplacian. The quantity L^+ has no clear intuitive meaning in terms of the structure of the graph G, and we could find no direct random walk equivalent. We found, however, that the third power P^3 of the transition matrix $P = D^{-1}A$ of a random walk was an equally effective measure P^3 has the straightforward interpretation of 'What Movies we expect a random walk to be at after 3 steps from a given start User'. (The walk is of the form: User-Movie-User-Movie). A full explanation of all methods is given in Section 8.2.1.

2. We based our experiments around repeated short random walks from a given start. The details of these experiments are given in Section 8.6. Our experiments worked quite well. We managed to match the performance of all methods from [50, 51] except the L^+ method, for which we could not find a short random walk equivalent. Results are given in Table 8.1.

   We have also computed the values of P^3 using matrix operations. In common with our findings that P^3 gave best results among most matrix based methods examined in [50, 51], we find that random walks of length 3 give best results. The details of the methods in Tables 8.1–8.3 are explained in Section 8.6.

   Carrying out the experiments highlighted statistical issues which required some thought. In parallel with our study we became aware of a study using a similar approach [120], but our conclusions are somewhat different. This is discussed further in Section 8.6

3. We evaluated the performance of the methods from [50, 51] as well as some additional methods using a larger dataset (MovieLens 1M), which was not used in [50, 51]. The main observation from this experiment is that the P^3 method has overtaken the L^+ method. We also found out that matrix based techniques did not scale well, as the matrix operations were now large. Techniques based on short random walks were shown to be more scalable and required significantly less memory than matrix operations.

8.4 Ranking algorithms

The input for the ranking methods is the bipartite graph $G = (U \cup I, R)$. The methods are general, but we describe them using the terminology of the MovieLens datasets, since the evaluation presented in [50,51] and our evaluation are based on those datasets. Thus we refer to U as the set of users, to I as the set of movies, and the edges in G show which movies the users have watched. A ranking algorithm computes for each user a ranking of movies. The main ranking methods considered in [50, 51] are described below.

Degree based ranking (MaxF): Movies that the user has not watched are ranked based on their degree in G. The highest degree movies (movies watched by the largest number of people) are at the top of this ranking.

Hitting time ranking (Hit$^{\rightarrow}$): Movies that the user has not watched are ranked based the hitting times of the random walk in G starting from that user. Movies with lower hitting times are higher up in the ranking.

Reverse hitting time ranking (Hit$^{\leftarrow}$): Movies that the user u has not watched are ranked based on the hitting times of the random walks in G starting from the movies. If the hitting time of u for the random walk starting at a movie m' is lower than the hitting time for the random walk starting from a movie m'', then m' is higher in the ranking than m''.

Commute time ranking (AVC): Movies that the user has not watched are ranked based the random walk commute time between that user and the movies.

Pseudoinverse Laplacian ranking (L^+): Let L^+ denote the Moore-Penrose pseudoinverse of the Laplacian matrix $L = D - A$ [109, 115]. Let $l^+_{x,y}$ denote the entry of L^+ at row x and column y. We compute L^+ using the following formula.

$$L^+ = (L - ee^T/n)^{-1} + ee^T/n, \tag{8.1}$$

where n is the number of nodes, e is a column vector made of 1s, i.e, $e = [1, 1, \ldots, 1]^T$.

Movies that the user u has not watched are ranked according to the descending order of the values $l^+_{u,m}$, $m \in I$.

Fouss *et al.* [50, 51] applied the above ranking methods to the small 100K MovieLens dataset using Metric I to evaluate their performance. They reported that the simple degree based ranking performed reasonably well, scoring 85%, which is significantly above the 50% score of the random permutation. The degree based ranking is probably the simplest possible method based fully only on the popularity of individual movies and requires little computation. The hitting time based rankings are considerably more costly to compute, but they did not perform better than the degree based ranking. The hitting time ranking could only match the performance of the degree based ranking, while the reverse hitting time ranking performed actually worse, scoring only 80%. The best ranking method was the L^+ ranking, scoring 90%.

Another widely used ranking algorithm is the ItemRank (IR) method described in [59], and the following methods.

ItemRank

The IR algorithm [59] is a random walk based scoring algorithm, which exploits correlations between items to predict user preferences. Recommendation is made based on the information obtained. Let $\mathcal{M}$ be the $|I| \times |I|$ correlation matrix, in which each element $\mathcal{M}_{ij}$ represents the number of users who have watched both movies m_i and m_j in the training set. Let $\tilde{\mathcal{M}}$ be the column normalised matrix of $\mathcal{M}$ and directed graph G_c be the correlation graph associated to the normalized correlation matrix $\tilde{\mathcal{M}}$. The IR algorithm is based on the personal PageRank algorithm [62].

$$\mathbf{IR} = \alpha \cdot \tilde{\mathcal{M}} \cdot \mathbf{IR} + (1 - \alpha) \cdot \tilde{\mathbf{d}}, \tag{8.2}$$

where α is attenuation factor typically 0.85 and $\tilde{\mathbf{d}}$ is the column normalized matrix of rating distribution matrix $\mathbf{d}$. The column vector $\mathbf{d_i}$ of $\mathbf{d}$ contains the user i' ratings on movies. Note that for unweighed case, the column vector $\mathbf{d_i}$ of $\mathbf{d}$ has a value of either 0 (not watched) or 1 (has watched) instead of user ratings (1-5) on movies.

For user i, among the unrated items, the items with the higher probability are recommended to the user i.

In addition to the above ranking algorithms, we experimented with two additional ranking methods, the s-step random walk distribution P^s method, the number of paths of length s method A^s and the parametrised s-step random walk distribution method P^3_α.

The s-step random walk distribution ($\mathbf{P^s}$) :

Movies that the user u has not watched are ranked based on the probability distribution $P^s(u,.)$ of the random walk at step s, if u was the starting vertex. If $P^s(u,m') > P^s(u,m'')$, then movie m' is ranked higher than movie m''. The matrix P^s can be computed by raising the matrix P of the transition probabilities of the random walk on G to the power of s.

Observe that only odd numbers $s \geq 3$ should be considered: for a user u and a movie m not watched by u, the probability $P^s(u,m)$ can be positive only for an odd $s \geq 3$. In this paper we include experimental results only for the P^3 and P^5 rankings (the rankings based on the distribution of the random walk after 3 and 5 steps, respectively), since they performed the best among the P^s rankings.

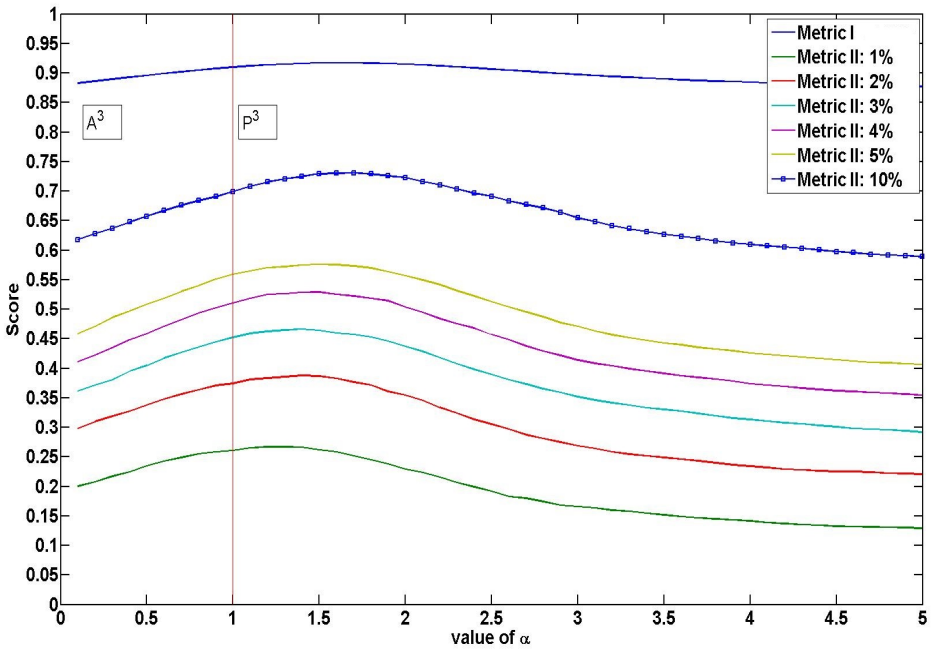
Number of paths of length s (A^s):

Movies that the user has not watched are ranked based on the number of distinct paths of length s from that user to the movies in graph G. A movie m with the greater number of paths has a higher ranking. The numbers of paths of length s can be simply computed as the s-th power of the adjacency matrix A.

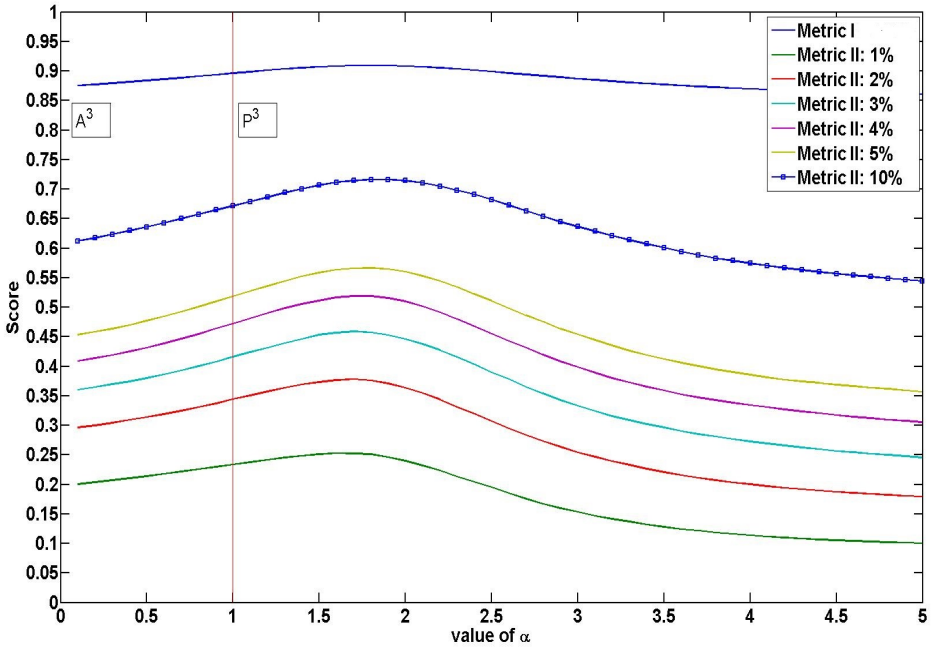
Parametrised method $\mathbf{P^s_\alpha}$

The method $\mathbf{P^s_\alpha}$ is a method for improving the performance of P^s by introducing a parameter α, which ranges over the real numbers. We define matrix P_α as derived from the transition probability matrix P by raising every entry to the power of α. That is,

$$p_\alpha(u,v) = \begin{cases} \left(\frac{1}{d(u)}\right)^\alpha, & \exists (u,v) \in E \\ 0, & \text{otherwise} \end{cases}$$



(a) Small Dataset



(b) Medium Dataset

Figure 8.1: Results of the P^3_α method using matrix operations. This incorporates the results of the A^3 method when $\alpha = 0$ and the P^3 method when $\alpha = 1$.

The parametrised P^s_α matrix is a generalization of the methods P^s and A^s. Setting $\alpha = 0$ this method is equivalent to the A^s method while setting $\alpha = 1$ it is equivalent to P^s.

Experimentally, limiting the number of steps to 3 has shown to have the best performance. The P^3 and P^3_α methods compute recommendations for a user u based only on the neighbourhood of u in graph G of diameter 3. A closer look at the general structure of graph G can give some justification why these methods perform so well. The diameter of G is small – for example, the training set subgraph for the small MovieLens dataset has diameter 6, and on average 70% of the users are at distance 2 from a given user. Another evident characteristic of the MovieLens datasets is their relatively high density. For example, the average degree of a user in the training set of the small MovieLens dataset is 95. Thus another interesting question is how the ranking algorithms "scale down" when the density of the dataset decreases. The results of the three methods P^3, A^3 and P^3_α can be seen in Figure 8.1.

8.5 Matrix based operations: Implementation details

In this section, we discuss the technical issues that arise when implementing the ranking methods discussed above.

The values of the hitting time ($Hit^{\rightarrow}$), reverse hitting time ($Hit^{\leftarrow}$) and commute time (AVC) between all pair of nodes can be obtained from hitting time matrix. Fouss *et al.* in [50, 51] suggest a method to compute such hitting time matrix through the pseudo-inverse of the Laplacian matrix (L^+). However, this method has a high computational cost as it requires a presence of a pseudo-inverse of the Laplacian matrix, which involves computation of matrix inversion. Moreover, even if it was assumed that pseudo-inverse of Laplacian matrix already exists, the time complexity for computing the hitting time matrix from the pseudo-inverse of Laplacian matrix is $O(n^3)$, where n is the size of a Laplacian matrix. In order to compute the hitting time matrix H faster, a method presented in [95], could be used. Specifically,

$$X = (I - P + \pi)^{-1}(J - 2mD^{-1})$$
$$H = X - (e * \lambda^T) \tag{8.3}$$

where I is the identity matrix of size n, P is the transition probability matrix, π is the stationary distribution of matrix A, J is a matrix of ones of size n, m is the total number of edges in the bipartite graph, D is the diagonal matrix of vertex degrees, λ is a column vector containing a diagonals of X matrix, and e is a column vector of ones.

Based on Equation 8.3, we can compute the commute time matrix C as:

$$C = H + H^T$$

Matrix multiplication requires $O(n^2)$ space, therefore if the size of graph increases, computing P^s matrix may require unrealistic amount of memory space and hence becomes intractable. For example, in the large 10M dataset, there are $82,248$ nodes in total, therefore a naive approach of computing P^3 would need $\approx$ 150GB.

8.6 Ranking methods based on simulation of Random Walks

In order to obtain a ranking of movies, we would need to compute the ranking matrix using any of the methods described in Section 8.4. There are two ways to achieve this:

1. Compute the exact values of each movie in order to determine their relative ranking. This can be done with matrix operations. E.g. we can compute the hitting time matrix H from Equation 8.3 and based on this result we can compute the commute time matrix C. We could also compute P^s as the s-th power of the transition probability matrix P.

2. Approximate these values by simulating short random walks.

Ranking algorithms based on matrix manipulations have their natural limit: the sizes of the matrices may quickly become too large to fit in the computer memory. Even if

the graph G modelling the dataset is sparse, matrix operations involving the adjacency matrix can easily lead to dense matrices. For example, the pseudoinverse matrix L^+ and the hitting time matrix H are both dense. While we would not have problems with running the ranking algorithms from the previous section on the small MovieLens dataset, some technical issues of insufficient memory would start appearing when we move on to the medium size dataset (the MovieLens 1M set), and became a major obstacle for computing matrices in the largest MovieLens dataset. While we could have used external memory algorithms in order to solve these memory issues, this would have added significant delays caused by frequent access to the slower secondary storage. 0

We use the alternative approach, which is to gather information about the structure of a graph from simulations of random walks in this graph. For example, the hitting times used in the ranking methods in the previous section can be estimated by simulating random walks. Such simulations require only $O(m)$ space, to store the adjacency lists of the graph, with $m \propto n$ for sparse graphs (n is the number of vertices and m is the number of edges). Furthermore, the computation can be organised in a distributed manner, if the graph is scattered over a network.

Experiment Description

In order to estimate random walk properties without using matrix operations we made use of many short random walks of various lengths. The experiments were ran on a personal computer with an Intel Quad core CPU at 3 GHz (specifically Intel Core 2 Quad Q9650) and 8 GB DDR2-800 memory. We developed our implementations using the C# programming language.

The training and test set graphs were loaded into memory as adjacency lists. There were W random walks performed, each of length s. Each individual walk visited a number of vertices within s steps of the starting user. For each of these vertices, if they corresponded to movies, we recorded the number of times they were visited by the walks at a specific step. All movies were evaluated based on properties such as Truncated

hitting times, number of Hits, number of hits at step s and then ranked accordingly. The ranking was done using radix sort with two radices, the first being the property in which we evaluated against, and the second being a random number. This resulted in all movies to be ranked according to the evaluated property. In the case that two (or more) movies had the same score, they were ranked randomly. This was done to ensure that the ranking only indicated the performance of the methods being checked and not any secondary characteristics. The ranking obtained through each method was evaluated using both metrics described in Section 8.2.1.

For most experiments the values of s were $s = 3, 5, 7, 9$. The values of W varied according to s based on the budget/step allowance of each experiment. For the results in Tables 8.1-8.3 the budget was set to be the number of vertices in the graph n. We note how walks of length $s = 7, 9$ had diminishing scores.

Assuming that a training and test set were in memory, the recommendation system consisted of 3 components:

1. The movie assessor, which, for each user u and given a movie m, would yield the value of that movie according to the assessment method. There were various methods used which are described in e.g. Table 8.1 and this is initialized by running W random walks of length s starting from the user u.

2. The item ranker, which uses a variety of different sort algorithms to rank items according to the assessment score. We made use of both the built-in $C\#$ quick-sort algorithm as well as our implementation of radix sort to perform the ranking.

3. The score evaluator which, given a ranked item list and the test set for a specific user, would return the score of that ranked list according to the various evaluation metrics defined in Section 8.2.1

Due to the independent nature of the process of evaluating each user's score we made use of multi-threading to a great extent and there were scores for multiple users being

computed concurrently. The drawback of this approach was that in the case of the large data-set, keeping all the assessment details in memory for many users was not feasible and we had to impose a limit of 4 concurrent threads. In practice having more threads is not expected to improve the performance of the algorithms since typical home computers don't currently have more than 4 processors.

Experimental Results

Recall that our ranking methods based on simulating random walks perform for each user n/s random walks, each of length s. We experimented with various values of s and found out that small values (that is, many short random walks) give the best results. We chose $s = 3, 5, 7, 9$ to compare the performance. In Table 8.4 we compare the metric-II scores of our ranking methods, counting the fraction of hits in the top k% recommendations.

The method $\hat{P}^3$ has the best performance. This method would converge to the method P^3 considered in Section 8.4, if the number of walks was increasing. The method based on the number of hits weighted with the vertex degrees comes second. Among the methods based on the truncated hitting times, the penalties $\hat{m}$ and $2\hat{m}$/degree, combined with s equal to 3 or 5, led to reasonably good performance.

Short Random Walk Evaluation Methodology

Similarly to the work of Sarkar and Moore [120], we estimate the *truncated hitting time* $h^T(u, v)$ of vertex v starting from vertex u by

$$\frac{1}{w} \sum_{i=1}^{w} s_v(i). \tag{8.4}$$

To use this estimator, we have to decide what should be the value of $s_v(i)$, if vertex v was not visited during walk i. Vertices which have not been visited by any of the walks are not ranked (or they are put in arbitrary order at the end of the ranking list, if a full ranking list is required), so we do not need to worry about their $s_v(i)$ values. We consider now a vertex v which was visited by some walks, but was not visited by, say,

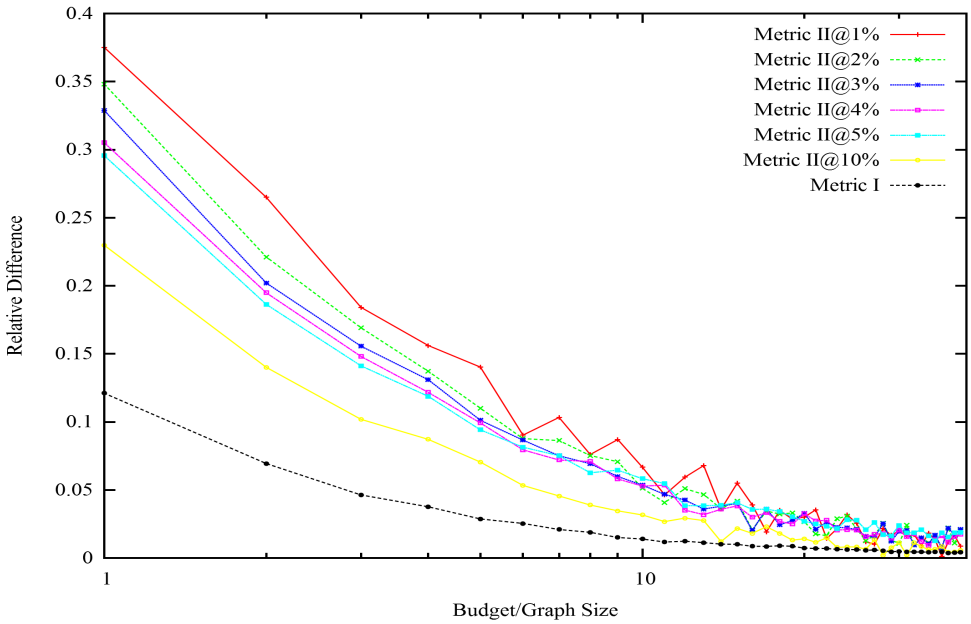


Figure 8.2: Convergence to P^3 on small dataset

walk i. A choice of $s_v(i) = 0$ seems incorrect, as this would imply $v = u$. Some hitting time penalty should be imposed for unvisited vertices, and we experimented with various values for this penalty.

- (a) Set the penalty to $\hat{m}$, our estimate of the number of edges incident with the subgraph we can visit in s steps.
- (b) Set the penalty to $2\hat{m}/d(v)$ where $\hat{m}$ is the estimated number of edges. The reasoning behind this penalty is to approximate the *first hitting time from stationarity* of a vertex. We do not use the total number of edges of a graph to cover the cases when this is unknown. The value $\hat{m}$ is our best estimate of the size of the sub-graph we are expected to see within s steps, and approximates the number of edges in the breadth first search tree of depth $s + 1$
- (c) Set the penalty to s, the walk length. This is the penalty used in [120].

Having estimated $h^T(u, v)$ with (8.4) and using one of the above penalties, we can rank the movies in ascending order of their $h^T(u, v)$ values.

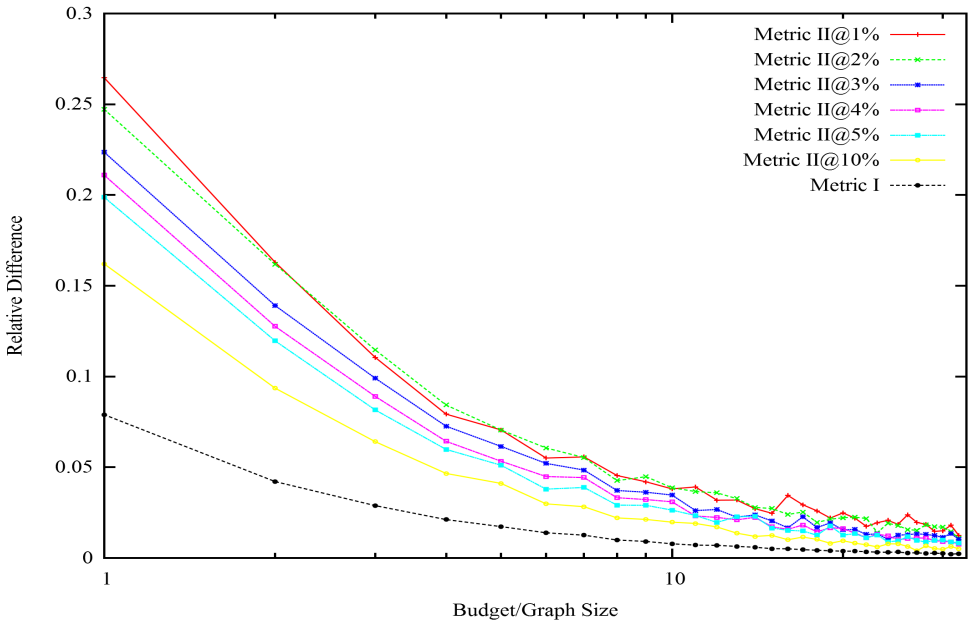


Figure 8.3: Convergence to P^3 on medium dataset

In addition to estimates of $h^T(u, v)$ we rank movies based on the number of times they were hit on average. We calculate the average number of hits of a vertex v from vertex u as

$$n(u, v) = \frac{1}{w} \sum_{i=1}^{w} n_v(i).$$

For example, if a movie m is hit $w/2$ times by w walks, then the number of hits is 0.5. Intuitively, movies which were hit often are more relevant to a user and should be ranked higher. Movies are ranked for recommendation in the descending order according to their average number of hits. We also consider some heuristics for re-weighting of average number of hits, such as (number of hits)*(degree).

We can also rank movies based on an estimate of the s step transition probability. We can estimate this simply by counting $\hat{P}^{(k)}(u, v)$ – the number of walks which hit vertex v at step s divided by the total number of walks w. If we let the number of walks increase to infinity, $\hat{P}^{(s)}(u, v)$ converges to $P^s(u, v)$. We rank movies in the descending order of values $\hat{P}^{(s)}(u, v)$.

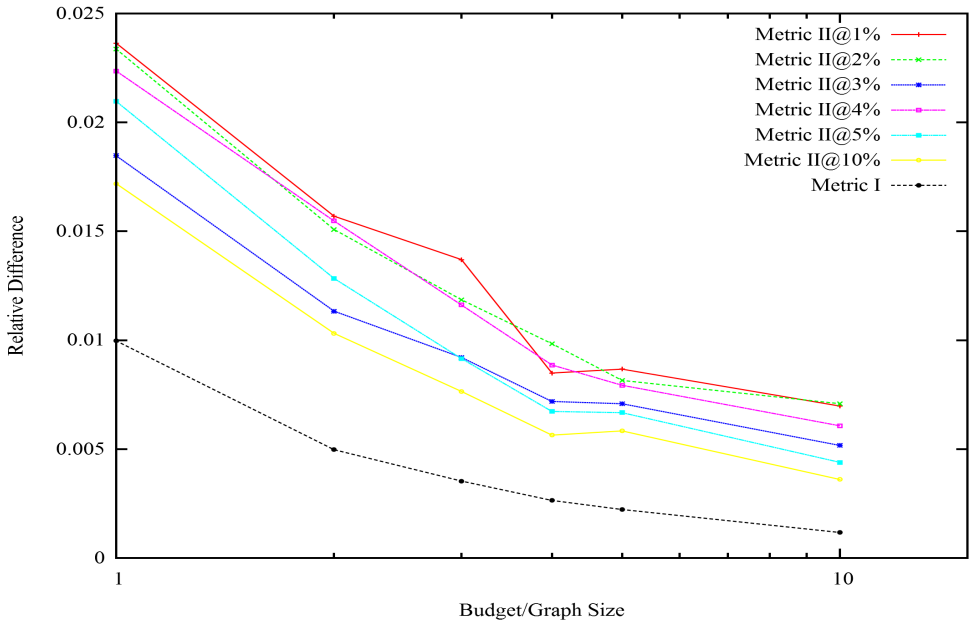


Figure 8.4: Convergence to P^3 on large dataset

Experimental results for SRW methods

Recall that our ranking methods based on simulating random walks perform for each user n/s random walks, each of length s. We experimented with various values of s and found out that small values (that is, many short random walks) give the best results.

The method $\hat{P}^3$ has the best performance. This method would converge to the method P^3 considered in Section 8.2.1, if the number of walks was increasing. The method based on the number of hits weighted with the vertex degrees comes second. Among the methods based on the truncated hitting times, the penalties $\hat{m}$ and $2\hat{m}/\text{degree}$, combined with s equal to 3 or 5, led to reasonably good performance. It would be of independent interest to investigate if these two methods give actually good estimators of the hitting times for these type of graphs.

Small size dataset							
Length of Walk (s)	Evaluation Method	**Truncated hitting times**			**#Hits**	**#Hits * degree**	$\hat{\mathbf{P}}^{\mathbf{s}}$
		penalty: Edges $\hat{r}$	penalty: $2\hat{r}$/degree	penalty: Walk length			
3	Metric I	78.98%	79.64%	79.08%	79.17%	80.45%	79.02%
	Metric II@1%	0.1642	0.1592	0.1621	0.1607	0.1760	0.1617
	Metric II@2%	0.2468	0.2297	0.2501	0.2477	0.2636	0.2477
	Metric II@3%	0.3034	0.2962	0.3056	0.3029	0.3300	0.3028
	Metric II@4%	0.3513	0.3441	0.3516	0.3496	0.3822	0.3514
	Metric II@5%	0.3903	0.3896	0.3885	0.3887	0.4286	0.3905
	Metric II@10%	0.5382	0.5545	0.5376	0.5389	0.5812	0.5356
5	Metric I	79.44%	80.12%	79.31%	79.33%	80.47%	70.88%
	Metric II@1%	0.1501	0.1602	0.1472	0.1474	0.1657	0.0905
	Metric II@2%	0.2304	0.2270	0.2310	0.2274	0.2504	0.1509
	Metric II@3%	0.2906	0.2948	0.2916	0.2877	0.3123	0.1947
	Metric II@4%	0.3435	0.3417	0.3410	0.3336	0.3644	0.2370
	Metric II@5%	0.3830	0.3819	0.3780	0.3766	0.4113	0.2755
	Metric II@10%	0.5319	0.5410	0.5253	0.5232	0.5723	0.3959
7	Metric I	79.20%	80.11%	78.78%	78.69%	80.16%	65.92%
	Metric II@1%	0.1462	0.1601	0.1454	0.1411	0.1615	0.0730
	Metric II@2%	0.2245	0.2275	0.2226	0.2217	0.2432	0.1208
	Metric II@3%	0.2785	0.2948	0.2769	0.2739	0.3046	0.1665
	Metric II@4%	0.3252	0.3407	0.3247	0.3156	0.3538	0.2000
	Metric II@5%	0.3642	0.3834	0.3609	0.3511	0.3962	0.2247
	Metric II@10%	0.5159	0.5400	0.5054	0.5001	0.5599	0.3353
9	Metric I	78.51%	79.86%	78.28%	78.07%	79.69%	63.47%
	Metric II@1%	0.1327	0.1601	0.1312	0.1287	0.1548	0.0586
	Metric II@2%	0.2060	0.2277	0.2084	0.2002	0.2317	0.1083
	Metric II@3%	0.2655	0.2932	0.2662	0.2572	0.2931	0.1407
	Metric II@4%	0.3133	0.3391	0.3117	0.3017	0.3436	0.1663
	Metric II@5%	0.3576	0.3803	0.3498	0.3438	0.3895	0.1930
	Metric II@10%	0.5013	0.5386	0.4910	0.4852	0.5478	0.3201

Table 8.1: Short Random Walks On the Small Dataset

Medium size dataset							
Length of Walk (s)	Evaluation Method	**Truncated hitting times**			**#Hits**	**#Hits * degree**	$\hat{\mathbf{P}}^{\mathbf{s}}$
		penalty: Edges $\hat{r}$	penalty: $2\hat{r}$/degree	penalty: Walk length			
3	Metric I	82.15%	82.80%	82.14%	82.12%	83.23%	82.10%
	Metric II@1%	0.1729	0.1729	0.1728	0.1834	0.1727	0.1731
	Metric II@2%	0.2583	0.2536	0.2587	0.2726	0.2584	0.2588
	Metric II@3%	0.3214	0.3203	0.3208	0.3368	0.3206	0.3213
	Metric II@4%	0.3716	0.3681	0.3714	0.3893	0.3715	0.3711
	Metric II@5%	0.4131	0.4102	0.4142	0.4344	0.4139	0.4143
	Metric II@10%	0.5620	0.5688	0.5621	0.5893	0.5617	0.5618
5	Metric I	81.92%	82.97%	81.74%	81.72%	83.09%	74.89%
	Metric II@1%	0.1600	0.1729	0.1597	0.1751	0.1581	0.1124
	Metric II@2%	0.2431	0.2533	0.2436	0.2600	0.2397	0.1789
	Metric II@3%	0.3043	0.3196	0.3036	0.3214	0.3002	0.2269
	Metric II@4%	0.3545	0.3670	0.3528	0.3732	0.3494	0.2687
	Metric II@5%	0.3947	0.4093	0.3946	0.4163	0.3908	0.3068
	Metric II@10%	0.5450	0.5673	0.5389	0.5741	0.5358	0.4427
7	Metric I	81.51%	82.87%	81.36%	81.32%	82.86%	71.26%
	Metric II@1%	0.1510	0.1730	0.1510	0.1703	0.1492	0.0956
	Metric II@2%	0.2324	0.2531	0.2323	0.2548	0.2278	0.1530
	Metric II@3%	0.2913	0.3195	0.2913	0.3160	0.2867	0.2004
	Metric II@4%	0.3408	0.3674	0.3407	0.3667	0.3364	0.2358
	Metric II@5%	0.3846	0.4096	0.3818	0.4089	0.3799	0.2702
	Metric II@10%	0.5345	0.5670	0.5284	0.5672	0.5268	0.3956
9	Metric I	81.18%	82.72%	80.90%	80.97%	82.59%	68.93%
	Metric II@1%	0.1476	0.1728	0.1469	0.1687	0.1457	0.0872
	Metric II@2%	0.2274	0.2532	0.2266	0.2507	0.2236	0.1409
	Metric II@3%	0.2871	0.3193	0.2839	0.3121	0.2828	0.1803
	Metric II@4%	0.3334	0.3670	0.3323	0.3611	0.3309	0.2198
	Metric II@5%	0.3754	0.4090	0.3729	0.4044	0.3711	0.2556
	Metric II@10%	0.5222	0.5665	0.5169	0.5609	0.5179	0.3610

Table 8.2: Short Random Walks On The Medium sized dataset

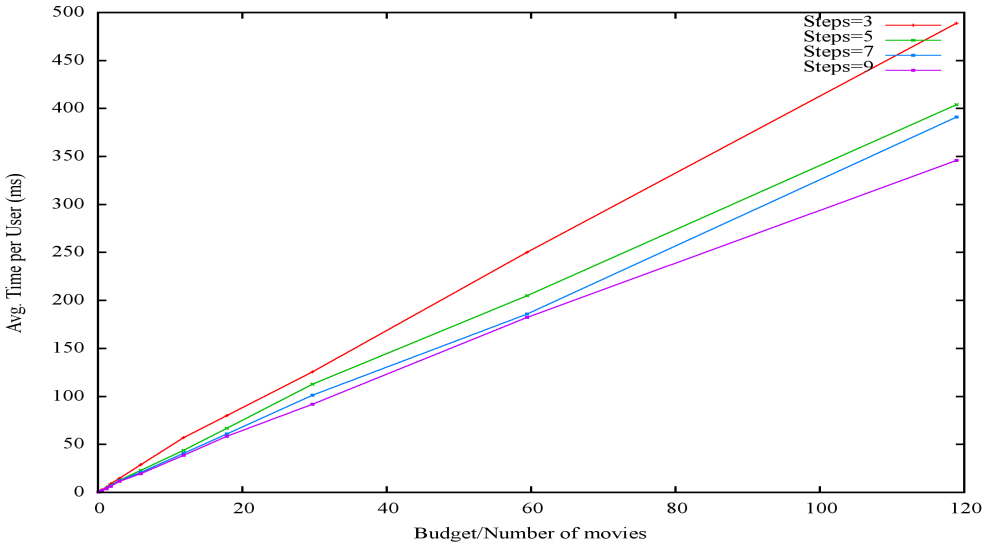


Figure 8.5: Average walk time for budget per user

8.6.1 Time measurements

In order to determine the feasibility of our approach using short random walks we have measured the time a short random walk requires to perform the necessary processing in order to produce user recommendations. What is seen in Figure 8.5 is an indication that the time required grows linearly with the total number of steps taken. Combining this with the fact that there is no pre-processing required and the methods can be ran for a single user in linear time we obtain a method that is more space and time efficient in the long run, compared to matrix methods.

8.6.2 Convergence of 3-step random walks to P^3

In this section we show experimental results obtained from varying the budget of the short random walk to $b = cn, \quad c \in \mathbb{N}$. What was done for this experiment is the same as was done to obtain the results of Tables 8.1-8.3 but specifically to estimate P^s with $s = 3$. These results were then compared to the results obtained using the matrix multiplication to obtain P^3 (see Table 8.4).

Large size dataset							
Length of Walk (s)	Evaluation Method	**Truncated hitting times**			**#Hits**	**#Hits * degree**	$\hat{\mathbf{P}}^{\mathbf{s}}$
		penalty: Edges $\hat{r}$	penalty: $2\hat{r}$/degree	penalty: Walk length			
3	Metric I	91.83%	93.65%	91.84%	94.90%	94.52%	94.90%
	Metric II@1%	0.1769	0.3507	0.1769	0.4681	0.4210	0.4681
	Metric II@2%	0.3087	0.4775	0.3087	0.5868	0.5419	0.5865
	Metric II@3%	0.4074	0.5681	0.4074	0.6645	0.6206	0.6644
	Metric II@4%	0.4855	0.6330	0.4855	0.7186	0.6828	0.7190
	Metric II@5%	0.5471	0.6766	0.5471	0.7590	0.7281	0.7585
	Metric II@10%	0.7461	0.8117	0.7461	0.8657	0.8471	0.8655
5	Metric I	91.04%	93.55%	91.06%	94.37%	94.15%	92.30%
	Metric II@1%	0.1495	0.3496	0.1495	0.4347	0.3985	0.3823
	Metric II@2%	0.2725	0.4763	0.2725	0.5535	0.5232	0.5035
	Metric II@3%	0.3693	0.5675	0.3693	0.6333	0.6022	0.5821
	Metric II@4%	0.4473	0.6318	0.4473	0.6901	0.6641	0.6375
	Metric II@5%	0.5122	0.6751	0.5122	0.7315	0.7103	0.6813
	Metric II@10%	0.7157	0.8109	0.7157	0.8474	0.8349	0.8032
7	Metric I	90.64%	93.52%	90.66%	94.10%	93.97%	90.82%
	Metric II@1%	0.1396	0.3490	0.1396	0.4159	0.3849	0.3440
	Metric II@2%	0.2545	0.4755	0.2545	0.5354	0.5140	0.4681
	Metric II@3%	0.3477	0.5668	0.3477	0.6157	0.5932	0.5488
	Metric II@4%	0.4254	0.6312	0.4254	0.6737	0.6544	0.6063
	Metric II@5%	0.4899	0.6746	0.4899	0.7169	0.7011	0.6503
	Metric II@10%	0.7004	0.8105	0.7004	0.8374	0.8290	0.7780
9	Metric I	90.35%	93.49%	90.34%	93.87%	93.83%	89.84%
	Metric II@1%	0.1319	0.3487	0.1319	0.4046	0.3789	0.3252
	Metric II@2%	0.2415	0.4754	0.2415	0.5239	0.5063	0.4491
	Metric II@3%	0.3338	0.5664	0.3338	0.6039	0.5868	0.5288
	Metric II@4%	0.4108	0.6311	0.4108	0.6624	0.6480	0.5883
	Metric II@5%	0.4753	0.6746	0.4753	0.7072	0.6947	0.6328
	Metric II@10%	0.6905	0.8105	0.6905	0.8308	0.8246	0.7623

Table 8.3: Short random walks on the large dataset

Dataset	Metric	Methods considered			
		$\hat{\mathbf{P}}^{\mathbf{3}}$	$\mathbf{P}^{\mathbf{3}}$	$\hat{\mathbf{P}}^{\mathbf{5}}$	$\mathbf{P}^{\mathbf{5}}$
Small	Metric I	80.45%	90.70%	80.47%	88.19%
	Metric II@1%	0.1617	0.261	0.0905	0.197
	Metric II@2%	0.2477	0.375	0.1509	0.295
	Metric II@3%	0.3028	0.452	0.1947	0.362
	Metric II@4%	0.3514	0.510	0.2370	0.415
	Metric II@5%	0.3905	0.559	0.2755	0.460
	Metric II@10%	0.5356	0.699	0.3959	0.616
Medium	Metric I	82.10%	89.62%	74.89%	86.68%
	Metric II@1%	0.1731	0.2336	0.1124	0.1809
	Metric II@2%	0.2588	0.3441	0.1789	0.2699
	Metric II@3%	0.3213	0.4160	0.2269	0.3332
	Metric II@4%	0.3711	0.4723	0.2687	0.3820
	Metric II@5%	0.4143	0.5183	0.3068	0.4250
	Metric II@10%	0.5618	0.6713	0.4427	0.5857
Large	Metric I	94.90%%	95.88%	92.30%	94.52%
	Metric II@1%	0.4681	0.4809	0.3823	0.4054
	Metric II@2%	0.5865	0.6027	0.5035	0.5270
	Metric II@3%	0.6644	0.6804	0.5821	0.6044
	Metric II@4%	0.7190	0.7376	0.6375	0.6667
	Metric II@5%	0.7585	0.7778	0.6813	0.7136
	Metric II@10%	0.8655	0.8845	0.8032	0.8390

Table 8.4: Comparison of the scores of P^3 and P^5 and the estimates $\hat{P}^3$ and $\hat{P}^5$ obtained through short random walks.

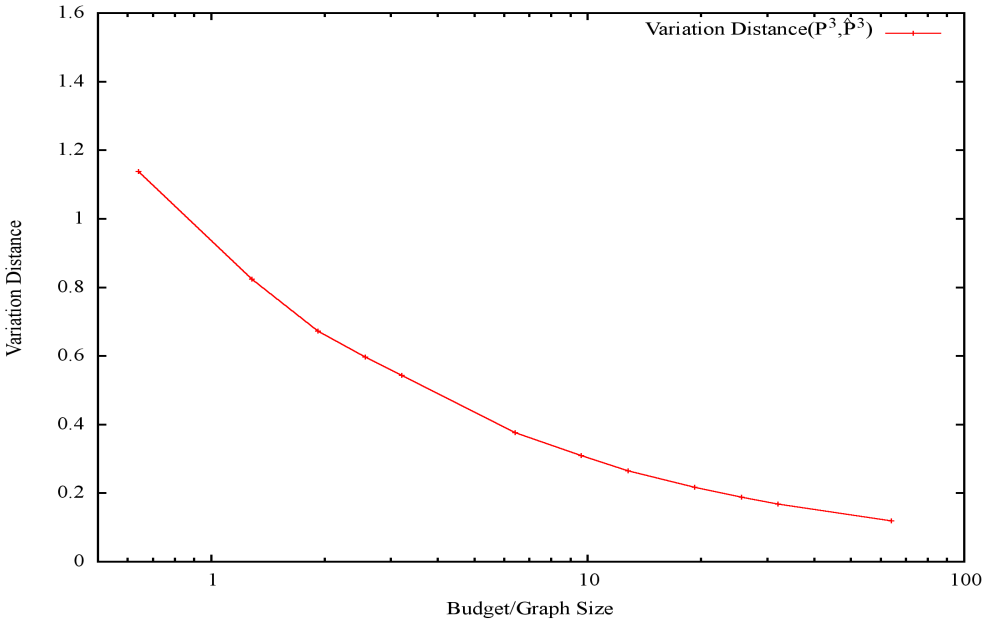


Figure 8.6: Total variation distance

The comparison was done using the formula:

$$\text{Relative difference} = \frac{|\text{Score of } \hat{P}^3 - \text{Score of } P^3|}{\text{Score of } P^3} \tag{8.5}$$

where $\hat{P}^3$ is the estimated P^3 obtained via short random walks while P^3 is the matrix obtained from matrix multiplication.

The results for the small dataset can be seen in Figure 8.2, for the medium dataset in Figure 8.3 and for the large dataset in Figure 8.4, where the relative difference is the value obtained by Equation (8.5). The x-axis is on a log-scale to emphasize how the increase of the budget greatly diminishes the benefit on the score. It is made especially apparent in the case of the large dataset (Figure 8.4) that the budget required to get a low relative difference is sub-linear in the size of the graph.

Additionally, in Figures 8.2,8.3,8.4 convergence is assumed to be achieved when the short random walk ranking yields the same evaluation score as the matrix operation score. However, we also want to confirm that the short random walk estimations of $\widehat{p}^3(u,m)$ (where $u \in U$ and $m \in I$) indeed converge to the real $p^3(u,m)$ entry of P^3.

This is done using the following a form of *total variation distance*:

$$V(P^3) = ||P^3 - \hat{P}^3||_\infty \tag{8.6}$$

We note that the value range of Equation (8.6) is $[0 \dots 2]$ with 0 indicating identical distributions. The results of Equation (8.6) applied to the small dataset can be seen in Figure 8.6.

8.6.3 Improving recommendations using degree-weighted Random Walks

The transition probabilities of a weighted random walk in a graph $G = (V, E)$ are defined by the weights of edges. Let $w(u, v) = w(v, u)$ be the weight of an (undirected) edge (u, v). The weight of a vertex u is defined as $w(u) = \sum_{(u,x)\in E} w(u, x)$. If the random walk is at a vertex u at the beginning of the current step, then the probability that it moves in this step to a vertex v is equal to

$$P(u, v) \;=\; \frac{w(u, v)}{w(u)}.$$

Ikeda *et al.* [67] consider the weighted random walks defined by the following edge weights:

$$w(u, v) = (d(u)d(v))^\beta\,, \tag{8.7}$$

where β is a parameter taking on real values. Observe that for this walk, the transition probability from the current vertex u to a neighbouring vertex v is equal to

$$P(u, v) \;=\; \frac{(d(u)d(v))^\beta}{\sum_{(u,x)\in E} (d(u)d(x))^\beta} \;=\; \frac{(d(v))^\beta}{\sum_{(u,x)\in E} (d(x))^\beta} \;\sim\; (d(v))^\beta.$$

Thus, for $\beta < 0$, the probability of moving into a given neighbouring vertex is larger, if its degree is smaller, and the stationary probability of high degree vertices decreases (in comparison with the uniform random walk). A similar effect is achieved by the weighted

random walk defined by the edge weights

$$w(u,v) = (\max\{d(u), d(v)\})^{\beta}\,, \tag{8.8}$$

considered in Ikeda *et al.* [66], and by the edge weights

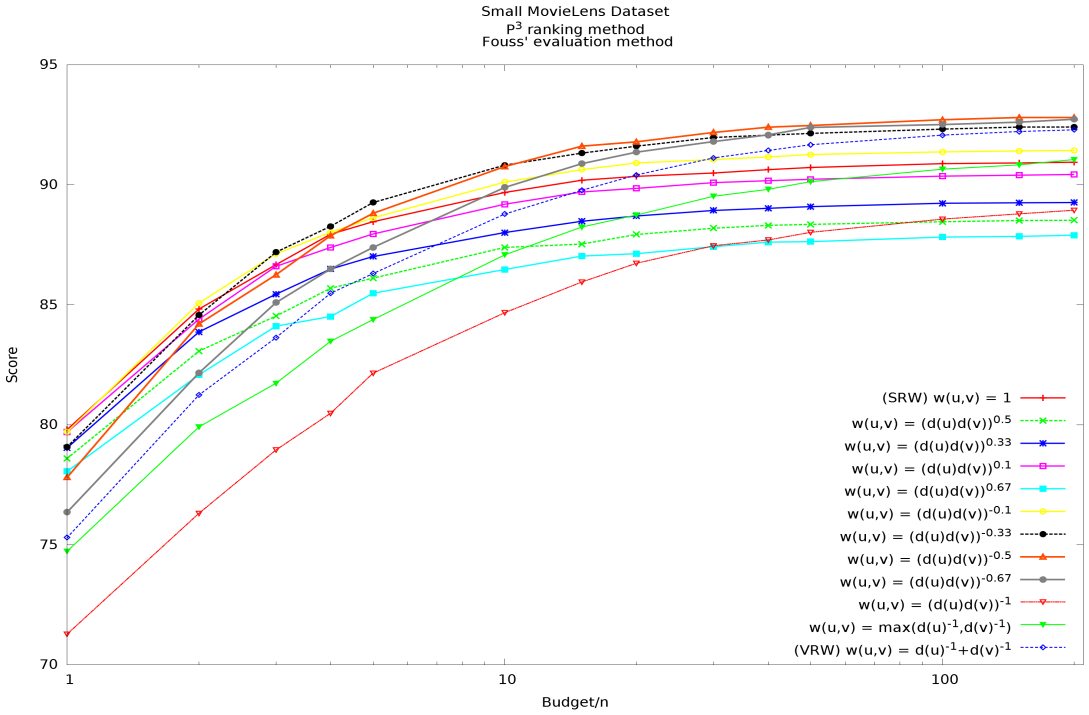
$$w(u,v) = (d(u))^{\beta} + (d(v))^{\beta}. \tag{8.9}$$

In both cases (8.8) and (8.9), if $\beta < 0$, then for two neighbours v' and v'' of u, $P(u,v') \leq P(u,v'')$, if $d(v') \geq d(v'')$. It is shown in [66, 67] that the cover time of the weighted random walk defined by (8.7) for $\beta = -1/2$ and the cover time of the weighted random walk defined by (8.8) for $\beta = -1$ are both $O(n^2 \log n)$ (while the worst-case cover time of the uniform random walk is $\Theta(n^3)$). Using a similar analysis as in [66, 67], one can also show that the cover time of the weighted random walk defined by (8.9) for $\beta = -1$ is also $O(n^2 \log n)$.

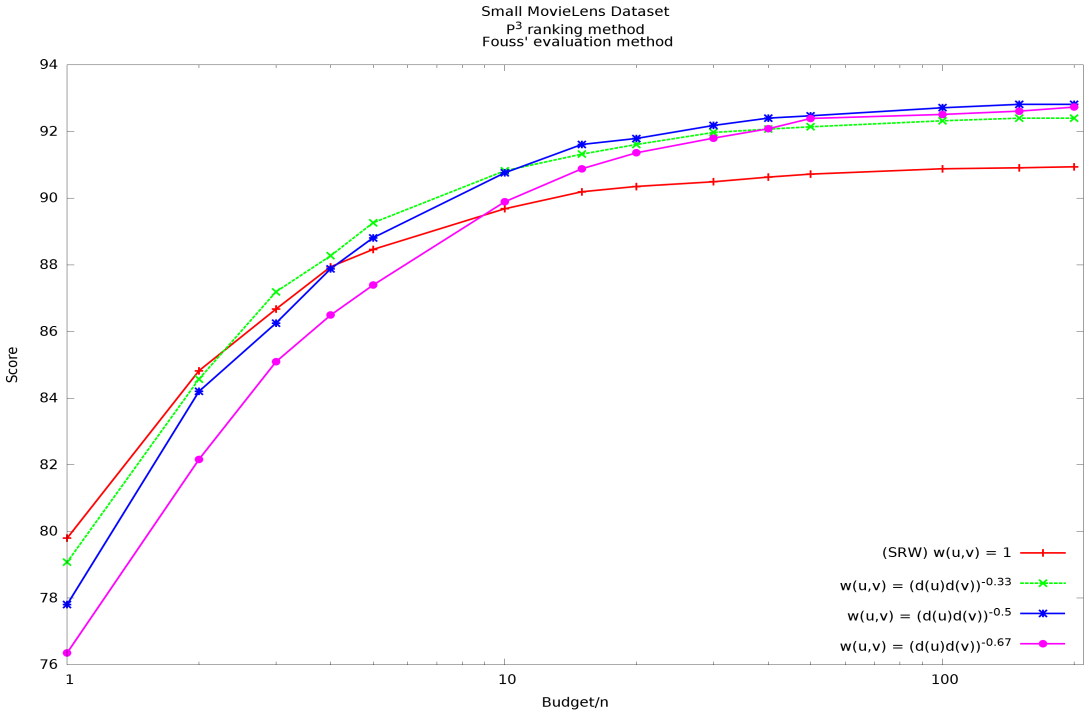
For the weighted random walks defined by (8.7), (8.8) and (8.9), we consider the ranking algorithm $\widehat{P^3}$, which estimates for each pair of vertices $u \in U$ and $v \in I$ the probability that the random walk which started at u is at v after three steps. This estimation is obtained by simulating $B/3$ random walks of length 3 for each starting vertex u and counting how many times each vertex $v \in I$ has been reached. Vertices with the same count are ranked in random order. The parameter B is the budget: the total number of steps of all random walks from the same starting vertex.

8.7 Experimental Results

Figures 8.7 and 8.8 show the plots of the quality of the rankings computed by the $\widehat{P^3}$ method for the 100K MovieLens dataset. We used the weighted random walks defined by the edge weights (8.7) with β taking on various values between -1 and 1, the edge weights (8.8) with $\beta = -1$, and the edge weights (8.9) with $\beta = -1$. Figure 8.7 shows the quality of the rankings computed according to the correctly-placed-pairs measure, which we refer to also as Fouss' measure. Figure 8.8 shows the quality of the rankings computed

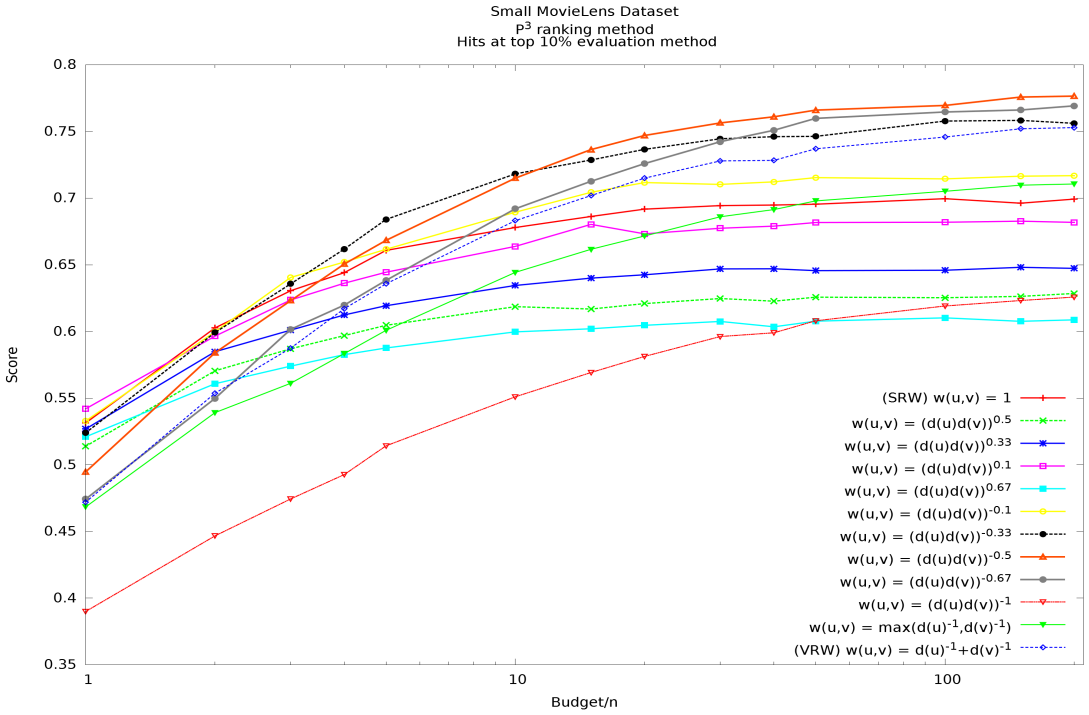


(a) Rankings of all tested methods

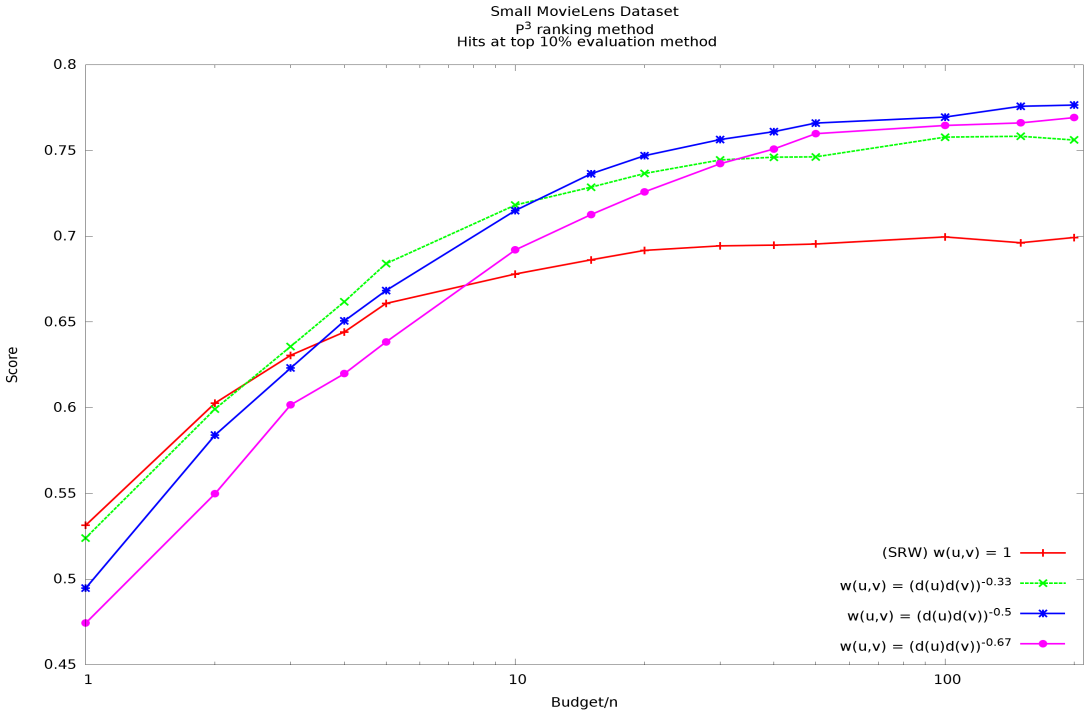


(b) Rankings of best methods compared to a SRW

Figure 8.7: The scores of the ranking algorithms based on various weighted random walks for the 100K MovieLens dataset: Fouss' measure.



(a) Rankings of all tested methods



(b) Rankings of best methods

Figure 8.8: The same experiments as in Figure 8.7, but the quality of rankings shown in the top 10% measure.

in the top 10% measure. In both figures, the left diagrams show the performance of all tested random walks. The right diagrams show only the random walks which give the best performance and the uniform random walk for comparison.

Interestingly, for the small budget $B = n$, rankings obtained by the weighted random walks are not better than the rankings obtained by the uniform random walks. However, when we let the budget increase, some of the weighted random walks start outperforming the uniform random walks, gaining clear advantage for $B = 10n$ and further improving until their performance flattens from $B \approx 100n$. The best performance is obtained for the weighted random walks defined by the edge weights (8.7) with $\beta = -1/2$ (the same β which gives the best bound for the worst-case cover time [67]). In comparison with the uniform random walks, the quality of the ranking obtained for $B = 100n$ increases from 91% to 93% in Fouss' measure, and from 0.70 to 0.77 in the top 10% measure. The weighted random walks defined by the edge weights (8.7) and $\beta = -1$ also eventually (for sufficiently large budget B) start outperforming the uniform walks.

8.8 Conclusion

In this chapter we have defined what we consider a recommender system. We view a recommender system as an algorithm on a graph $G = \{(U \cup I), E\}$ where U is a set of users, and I is a set of items those users could potentially interact with. When given a user $u \in U$, the purpose of a recommender algorithm is to rank items in I according to the user's relation to those items. Specifically a good recommender algorithm would place items which a user u is more likely to be interested in, higher in the ranking.

We build on the work of Fouss *et al.* [50, 51] by making use of RW based methods to produce ranked lists of I. We work with the dataset collected by GroupLens [60] of users and movies they had rated using the MovieLens online service [111]. We have shown that SRW methods such as the third power of the transition probability matrix P^3 work equally well, and sometimes better than the methods considered in [50, 51]. In addition, we generalize the method of P^3 by raising each entry of P to the power α. We call this

method P^3_α and we show how this method can be used to optimize the results of P^3. Furthermore we show how the method of P^3 in conjunction with degree biased WRWs could further improve the quality of the results. We test the quality of the results using two quality metrics: Metric I, percentage of correctly placed pairs and Metric II, number of hits in the top k recommendations.

Chapter 9

Conclusions

9.1 Scale-free graphs and generative models

In Chapter 2 we have seen a number of graph properties which appear in large online networks, as well as graph generation models which generate graphs which share these properties with large online networks. As mentioned, there are a number of properties which are indicative of large online networks, such as:

- Power-law degree distribution.
- Diameter bounded by $O(\log n)$.
- Existence of sub-graphs which are internally densely connected and externally sparsely connected. We refer to these sub-graphs as communities.
- Scale invariance i.e. the local structure and global properties are unaffected by the size of the graph.
- Higher than expected number of triangles.

We generally refer to graphs with these properties as *scale-free graphs*. There are many real world graphs which belong to the category of scale-free graphs such as: the router network, the WWW and OLSNs. There are various existing graph models which generate graphs which have similarities to these networks. Models such as the Erdős-Rényi models

for example generate sparse connected graphs with relatively small diameters, while the preferential attachment models (such as e.g. the Barabási-Albert model [4] or the general web-graph model [24, 28]) generate graphs which also have a degree distribution which follows a power-law. While these two properties are significant in real online networks, they are not sufficient to accurately model them, therefore additional models were proposed, such as e.g. the edge copying model [72], the triangle closing model [85] etc. We have also proposed a number of our own models such as:

1. Random walk graph model (Section 2.2.8).
2. Preferential attachment with message propagation model (Section 2.2.9).
3. Grow, back-connect, densify model (Section 2.2.10).
4. Partitioned preferential attachment model (Section 2.2.11).
5. Implicit Power-Law graph model (Section 2.2.12).

There are some open problems which remain unanswered in this area and ongoing research focuses on the following aspects:

1. Discovery of additional properties which are common among scale-free networks.
2. Creation of a graph generation model which more accurately models real online networks.
3. Formal analysis of graph generation models which would accurately describe the properties which generated graphs would have.

9.2 Markov Chains and Random Walks

In Chapter 3 we have presented an overview of Markov Chains and specifically, time-homogeneous Markov Chains on graphs which we refer to as Random Walks. We have discussed notions such as transition probability matrix $\mathbf{P}$ and stationary distribution

π as well as mixing rate τ_2 and mixing time $T(\epsilon)$. We base the properties of Markov Chains we have presented, from the books Norris [113] and by Aldous and Fill [2], as well as the survey by Lovasz [95].

Simulating RWs is a large part of our work and therefore we discuss the algorithms that could be used, as well as the algorithmic issues which arise from simulating RWs and specifically the increase in complexity when simulating Weighted Random Walks (WRWs). We also present a method to make optimal use of caching to speed up query time but at the cost of increased space complexity. We show how this *dynamic caching* method greatly improves performance of WRWs. In addition, we show that if the total number of steps taken remains significantly lower than the cover time of the graph, dynamic caching also outperforms pre-processing methods.

In the area of simulating RWs there are still some open problems, the most important, in our opinion is discovery of a method which would further improve the query time of specific WRWs without requiring additional memory. This could be achieved by approximation as well as knowledge of specific RW and graph properties.

9.3 Estimating network properties

In Chapter 4 we have presented some methods to estimate network properties. The methods presented were from various backgrounds such as zoology, anthropology and sociology. These methods mainly rely on sampling *uar* which is a simple and unbiased method to obtain samples from a set. However it is not always possible to obtain samples *uar* from large online networks. For example, getting a web-page from the WWW *uar* is impossible unless we have obtained the entire network beforehand. Due to this we present some RW based methods to simulate *uar* sampling while at the same time not requiring any parts of the network to be stored in memory. This can be done by using either a RWRW or a MHRW to unbias a SRW.

We have also partially sampled the Twitter OLSN and shown the differences between sampling *uar* and sampling by using a SRW to discover new vertices. Because a SRWs is

degree biased, we observed that we had obtained a larger number of high degree vertices than those acquired by sampling *uar*.

9.4 Estimating network properties using Random Walks

In Chapter 5 we have shown additional methods to estimate network properties. The methods presented are based on random walks and they are the following:

Method of first returns This method makes use of the first return time of a RW to estimate the total weight of the graph. By using this method and appropriately designed WRWs we are able to estimate a variety of graph properties such as: number of edges, number of vertices, number of triangles, number of arbitrary fixed subgraphs and clustering coefficient.

Method of the Cycle formula of regenerative processes (CFRP) This method is a generalization of the first return times method. It also relies on the first return time of a RW but in this case, rather than counting the number of steps until the first return, we keep track of a cumulative sum of a vertex valued function. By using appropriately selected functions we are able to estimate the same properties as mentioned above, but we can do so by using any underlying RW.

Method of the running totals The method of the running totals is similar to the method of the CFRP in that we use the same vertex valued functions. However in this case, rather than waiting for the first return, we rely on knowledge of the graph weight of the underlying RW to get a property estimate at each step of the walk.

For each of these methods we have investigated the convergence rate both theoretically and experimentally. We have experimented using these methods on manufactured graphs such as graphs generated using the Barabási-Albert model. In addition we have applied these methods on datasets of large online networks. The experimental results obtained can be found in Chapter 6.

Each of the aforementioned method has certain advantages and disadvantages.

- The method of first returns is simple and many theoretical results exist regarding the first return times of RWs. It has been observed that scale-free networks are good expanders which means that a SRW on these networks converges rapidly to stationarity, which in turn means the property estimates obtained using a SRW are generally accurate. However estimating properties other than the number of edges, requires the use of appropriately designed WRWs. In this case we have no knowledge of the rate of convergence to stationarity.

- By using the method of the CFRP we can avoid this issue by using a SRW to estimate a number of properties. The disadvantage with this method however is that the accuracy of the estimates, is heavily correlated to the convergence rate of the underlying walk. It may be the case that on some graphs, we may still find it better to use the CFRP while using a WRW as the underlying walk.

- The method of the running totals does not rely on the first return time. The accuracy of the estimated property depends on the rate of convergence to stationarity rather than the rate in which the experimentally obtained first return times converge to the theoretically expected return time $\mathbf{E}T_u^+$. The advantage of this method is that it is very accurate and previous work has shown that the error in the estimate drops exponentially with the number of steps. However this method requires knowledge of the graph weight of the underlying RW, which is not always a realistic assumption.

There are some open problems which remain unanswered in this area, including:

- Creating additional WRWs for the purpose of estimating different network properties.

- Determine bounds for the eigenvalue gap for each of the RWs based methods on scale-free graphs.

- Experimentally confirm the convergence bounds of all proposed methods.
- Make use of on-line convergence tactics to further optimize the required amount of samples required.
- Deploy the proposed methods on real online networks to show their applicability.

9.5 Fast discovery of high degree vertices

In Chapter 7 we have presented a method based on a degree biased WRW which covers the set of high degree vertices faster than a SRW. We have proven, both theoretically and experimentally, that this method works on graphs generated using the general web-graph model (described in Section 2.2.2). In addition, the graph cover time of the degree biased WRW does not significantly increase compared to that of a SRW. We have also shown experimentally that this method could be used as a heuristic to quickly discover high degree vertices on general scale-free graphs, such as large online networks.

It would be of further interest to further investigate the structure of large online networks, in order to more accurately determine the optimal parameters for the degree biased WRW. This would allow us to further improve the discovery rate of high degree vertices on such graphs.

9.6 Information filtering and recommendation

In Chapter 8 we have presented our contribution to the area of recommender systems. In our work we focus on recommender systems based on collaborative filtering and build upon the work of Fouss *et al.* [50, 51]. A recommendation system is a system which comprises of several components, but in our work we focus only on ranking algorithms which, given a user u, would rank all items I according to that user's predicted interest in them.

Our contributions to this area are:

- We show that the a simple approach based on the transition probability matrix of a RW outperforms many of the state-of-the-art recommendation algorithms.
- We suggest ways to optimize the quality of the recommendations which are produced based on on the transition probability matrix of a RW.
- We show how using different types of WRW further improves the quality of the recommendations generated.
- We further improve on the space efficiency of existing methods by simulating RWs instead of using exact matrix operations. This results in estimated results which rapidly converge to the expected value, require less physical memory, and are obtained without adding any additional time complexity to the algorithms.

There are several open problems in this area which we would focus our future research, such as:

- Theoretically and experimentally determine the bias introduced by currently used evaluation techniques.
- Further improve the accuracy of the results by using WRWs specifically designed for this purpose.
- Make use of meta-data to determine secondary user preferences. We could achieve this by including the secondary characteristics of items (e.g. the genre of movies or the director of movies), and we could rank the user's preference according to these characteristics.

Bibliography

[1] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. *J. ACM*, 56(4), 2009.

[2] David Aldous and James Allen Fill. Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/pub/users/aldous/RWG/book.html, 1995.

[3] Ricardo Baeza-yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: Better strategies than breadth-first for web page ordering. In *Proceedings of the 14th international conference on World Wide Web / Industrial and Practical Experience Track*, pages 864–872. ACM Press, 2005.

[4] A. Barabási and R. Albert. Emergence of scaling in random networks. In *Science*, number 5439 in Volume 286, pages 509–512, 1999.

[5] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, 281(14):69 – 77, 2000.

[6] Albert-László Barabási, Zoltán Dezső, Erzsébet Ravasz, and Zoltán Oltvai. Scale-free and hierarchical structures in complex networks. In *In Sitges Proceedings on Complex Networks*, pages 1–16. Springer-Verlag, 2003.

[7] L. A. Bassalygo and M. S. Pinsker. The complexity of an optimal non-bloking commutation scheme without reorganization. *Problemy Peredači Informacii*, 9(1):84–

87, 1973. Translated into English in Problems of Information Transmission, 9 (1974) 64–66. MR0327387 (48:5729).

[8] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab, April 2003.

[9] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In Jude W. Shavlik, editor, *ICML*, pages 46–54. Morgan Kaufmann, 1998.

[10] G. Blom, L. Holst, and D. Sandell. *Problems and Snapshots from the World of Probability*. Problems and Snapshots from the World of Probability. Springer New York, 1994.

[11] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[12] B. Bollobás and O. Riordan. *Handbook of Graphs and Networks: Mathematical results on scale-free graphs*, pages 1–32. S. Bornholdt, H. Schuster (eds), Wiley-VCH, 2002.

[13] Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. random structures and algorithms. *Random Structures and Algorithms*, 18:279–290, 2001.

[14] Bélaa Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24:5–34, January 2004.

[15] Mickey Brautbar and Michael Kearns. Local algorithms for finding interesting individuals in large networks. In *ICS*, pages 188–199, 2010.

[16] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure

in the web: experiments and models. In *Proceedings of the Ninth International World-Wide Web Conference (WWW9, Amsterdam, May 15 - 19, 2000 - Best Paper)*. Foretec Seminars, Inc. (of CD-ROM), Reston, VA, 2000.

[17] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[18] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, Washington DC, USA, May 2010.

[19] Douglas George Chapman. *Some properties of the hypergeometric distribution with applications to zoological sample censuses.* Berkeley, University of California Press, 1951.

[20] J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in Analysis*, pages 195–199, 1971.

[21] Maggie X. Cheng, editor. *Nano-Net - Third International ICST Conference, NanoNet 2008, Boston, MA, USA, September 14-16, 2008, Revised Selected Papers*, volume 3 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer, 2009.

[22] Kai-Min Chung, Henry Lam, Zhenming Liu, and Michael Mitzenmacher. Chernoff-hoeffding bounds for markov chains: Generalized and simplified. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 124–135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[23] C. Cooper. Classifying special interest groups in web graphs. In *RANDOM 2002: Randomization and Approximation Techniques in Computer Science*, pages 263–275, 2002.

[24] C. Cooper and A. Frieze. A general model web graphs. In *Random Structures and Algorithms*, pages 311–335, 2003.

[25] C. Cooper and A. Frieze. The cover time of the preferential attachment graphs. *Journal of Combinatorial Theory*, B(97):269–290, 2007.

[26] Colin Cooper. The age specific degree distribution of web-graphs. *Combinatorics Probability and Computing*, 15:637–661, 2006.

[27] Colin Cooper, Robert Elsässer, Hirotaka Ono, and Tomasz Radzik. Coalescing random walks and voting on graphs. *CoRR*, abs/1204.4106, 2012.

[28] Colin Cooper and Alan M. Frieze. A general model of undirected web graphs. In Friedhelm Meyer auf der Heide, editor, *ESA*, volume 2161 of *Lecture Notes in Computer Science*, pages 500–511. Springer, 2001.

[29] Colin Cooper and Alan M. Frieze. The cover time of random regular graphs. *SIAM J. Discrete Math.*, 18(4):728–740, 2005.

[30] Colin Cooper and Alan M. Frieze. Random walks on random graphs. In Cheng [21], pages 95–106.

[31] Colin Cooper and Alan M. Frieze. Random walks on random graphs. In Cheng [21], pages 95–106.

[32] Colin Cooper, Alan M. Frieze, and Tomasz Radzik. Multiple random walks in random regular graphs. *SIAM J. Discrete Math.*, 23(4):1738–1761, 2009.

[33] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Recommender systems based on random walks. Technical report, Department of Informatics, King's College London, August 2013.

[34] Colin Cooper, Sang-Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Random walks in recommender systems: exact computation and simulations. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *WWW (Companion Volume)*, pages 811–816. ACM, 2014.

[35] Colin Cooper, Tomasz Radzik, and Yiannis Siantos. Estimating network parameters using random walks. In *CASoN*, pages 33–40. IEEE, 2012.

[36] Colin Cooper, Tomasz Radzik, and Yiannis Siantos. A fast algorithm to find all high degree vertices in graphs with a power law degree sequence. In *WAW 2012 Proceedings*, volume 7323 of *LNCS*, pages 165–178. Springer, 2012.

[37] Colin Cooper, Tomasz Radzik, and Yiannis Siantos. A fast algorithm to find all high degree vertices in power law graphs. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 1007–1016, New York, NY, USA, 2012. ACM.

[38] Colin Cooper, Tomasz Radzik, and Yiannis Siantos. Fast low-cost estimation of network properties using random walks. In Anthony Bonato, Michael Mitzenmacher, and Pawe Praat, editors, *Algorithms and Models for the Web Graph*, volume 8305 of *Lecture Notes in Computer Science*, pages 130–143. Springer International Publishing, 2013.

[39] Colin Cooper, Tomasz Radzik, and Yiannis Siantos. Estimating network parameters using random walks. *Social Network Analysis and Mining*, 4(1), 2014.

[40] Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 570–578, Washington, DC, USA, 2007. IEEE Computer Society.

[41] Stephen Dill, Ravi Kumar, Kevin S. Mccurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Trans. Internet Technol.*, 2:205–223, August 2002.

[42] I. Dinwoodie. A probability inequality for the occupation measure of a reversible markov chain. *Ann. Appl. Probab.*, 5:37–43, 1995.

[43] Jozef Dodziuk. Difference equations, isoperimetric inequality and transience of certain random walks. *Transactions of the American Mathematical Society*, 284(2):pp. 787–794, 1984.

[44] M. Enachescu E. Drinea and M. Mitzenmacher. Technical report: Variations on random graph models for the web. Technical report, Harvard University, Department of Computer Science, 2001.

[45] P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[46] S. Ye F. Wu, J. Lang. Crawling online social graphs. In *2010 12th International Asia-Pacific Web Conference*, 2010.

[47] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262, New York, NY, USA, 1999. ACM.

[48] Abraham Flaxman, Alan Frieze, and Trevor Fenner. High degree vertices and eigenvalues in the preferential attachment graph. *Internet Mathematics*, 2(1):1–19, 2005.

[49] Abraham D. Flaxman and Juan Vera. Bias reduction in traceroute sampling - towards a more accurate map of the internet. In *WAW*, pages 1–15, 2007.

[50] François Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowl. Data Eng.*, 19(3):355–369, 2007.

[51] François Fouss, Alain Pirotte, and Marco Saerens. A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In Andrzej Skowron, Rakesh Agrawal, Michael Luck, Takahira Yam-

aguchi, Pierre Morizet-Mahoudeaux, Jiming Liu, and Ning Zhong, editors, *Web Intelligence*, pages 550–556. IEEE Computer Society, 2005.

[52] A. Ganesh, A. Kermarrec, E. Le Merrer, and L. Massouli. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20:267–278, 2007. 10.1007/s00446-007-0027-z.

[53] Andrew Gelman and Donald B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):pp. 457–472, 1992.

[54] Gephi. https://gephi.org/, 2013.

[55] John Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *IN BAYESIAN STATISTICS*, pages 169–193. University Press, 1992.

[56] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. In *Proc Natl Acad Sci U S A*, 2002.

[57] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. A walk in facebook: Uniform sampling of users in online social networks. *CoRR*, abs/0906.0060, 2009.

[58] Marco Gori and Augusto Pucci. Research paper recommender systems: A random-walk based approach. In *Web Intelligence*, pages 778–781. IEEE Computer Society, 2006.

[59] Marco Gori and Augusto Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In Manuela M. Veloso, editor, *IJCAI*, pages 2766–2771, 2007.

[60] Grouplens. http://www.grouplens.org/node/73.

[61] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[62] Taher H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526, 2002.

[63] Don W. Hayne. Two methods for estimating population from trapping records. *Journal of Mammalogy*, 30(4):pp. 399–411, 1949.

[64] Jing He, John E. Hopcroft, Hongyu Liang, Supasorn Suwajanakorn, and Liaoruo Wang. Detecting the structure of social networks using (α, β)-communities. In *Proceedings of the 8th International Workshop in Algorithms and Models for the Web Graph (WAW)*, pages 26–37, 2011.

[65] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.

[66] Satoshi Ikeda, Izumi Kubo, Norihiro Okumoto, and Masafumi Yamashita. Impact of Local Topological Information on Random Walks on Finite Graphs. In *Proceedings of the 32st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1054–1067, Washington DC, USA, 2003.

[67] Satoshi Ikeda, Izumi Kubo, and Masafumi Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410(1):94 – 100, 2009.

[68] J. Kleinberg J. Leskovec and M. Faloutsos. Graph evolution: Densification and shrinking diameters. In *Knowledge Discovery from Data*, Vol. 1, No. 1, Article 2, 2007.

[69] Anonymous ratings data from the jester online joke recommender system. http://goldberg.berkeley.edu/jester-data.

[70] Ana Justel, Daniel Pea, and Rubn Zamar. A multivariate kolmogorov-smirnov test of goodness of fit. *Statistics & Probability Letters*, 35(3):251 – 259, 1997.

[71] Liran Katzir, Edo Liberty, and Oren Somekh. Estimating sizes of social networks via biased sampling. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar,

M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *WWW*, pages 597–606. ACM, 2011.

[72] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models, and methods. In *Proceedings of the 5th annual international conference on Computing and combinatorics*, COCOON'99, Berlin, Heidelberg, 1999. Springer-Verlag.

[73] J. Komjathy and Y. Peres. Mixing and relaxation time for Random Walk on Wreath Product Graphs. *ArXiv e-prints*, August 2012.

[74] B. Krishnamurthy, P. Gill, and M. Arlitt. A few chirps about twitter. In *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks*, WOSN08, 2008.

[75] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Extracting large-scale knowledge bases from the web. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB*, pages 639–650. Morgan Kaufmann, 1999.

[76] Jérôme Kunegis and Stephan Schmidt. Collaborative filtering using electrical resistance network models. In Petra Perner, editor, *Industrial Conference on Data Mining*, volume 4597 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2007.

[77] Last.fm. http://www.last.fm/.

[78] Silvio Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 427–434, New York, NY, USA, 2009. ACM.

[79] Sangkeun Lee, Sang il Song, Minsuk Kahng, Dongjoo Lee, and Sang goo Lee. Random walk based entity ranking on graph for multidimensional recommenda-

tion. In Bamshad Mobasher, Robin D. Burke, Dietmar Jannach, and Gediminas Adomavicius, editors, *RecSys*, pages 93–100. ACM, 2011.

[80] Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang goo Lee. Pathrank: Ranking nodes on a heterogeneous graph for flexible hybrid recommender systems. *Expert Syst. Appl.*, 40(2):684–697, 2013.

[81] C. Léon and F. Perron. Optimal hoeffding bounds for discrete reversible markov chains. *The Annals of Applied Probability*, 14(2):958–970, 2004.

[82] Albert Leon-Garcia. *Probability and Random Processes for Electrical Engineering (2nd Edition)*. Addison-Wesley, 2 edition, August 1993.

[83] J. Leskovec. Stanford network analysis package. http://snap.stanford.edu/, 2009.

[84] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of of ACM SIGKDD*, ACM SIGKDD 2006, 2006.

[85] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 462–470, New York, NY, USA, 2008. ACM.

[86] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, March 2010.

[87] Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 497–504, New York, NY, USA, 2007. ACM.

[88] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1, March 2007.

[89] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008.

[90] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168, 1944.

[91] P. Lezaud. Chernoff-type bound for finite markov chains. *The Annals of Applied Probability*, 8(3):849–867, 1998.

[92] Lg u+ oz store. http://ozstore.uplus.co.kr/.

[93] Frederick C. Lincoln. Calculating waterfowl abundance on the basis of banding returns. Circular 118, United States Department of Agriculture, Washington D.C., May 1930.

[94] Livejournal. http://www.livejournal.com.

[95] László Lovász. Random walks on graphs: A survey. *Bolyai Society Mathematical Studies*, 2:353–397, 1996.

[96] Mohammad Mahdian and Ying Xu. Stochastic kronecker graphs. Technical report, Proceedings of the 5th Workshop on Algorithms and Models for the Web-Graph, 2007.

[97] Rob Malda and Jeff Bates. Slashdot. slashdot.org.

[98] George Marsaglia, Wai Wan Tsang, and Jingbo Wang. Evaluating kolmogorov's distribution. *Journal of Statistical Software*, 8(18):1–4, 11 2003.

[99] Gerald G. Marten. A regression method for mark-recapture estimation of population size with unequal catchability. *Ecology*, 51(2):pp. 291–295, 1970.

[100] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In Eric Ruppert and Dahlia Malkhi, editors, *PODC*, pages 123–132. ACM, 2006.

[101] Mathworks. Matlab. http://www.mathworks.co.uk/products/matlab/, 2014.

[102] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, 1998.

[103] E. H. Mckinney. Generalized birthday problem. *The American Mathematical Monthly*, 73(4):pp. 385–387, 1966.

[104] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal Of Chemical Physics*, 21(6):1087–1092, 1953.

[105] Microsoft. C#.net. http://www.microsoft.com/net, 2010.

[106] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert E. Tarjan. Finding strongly knit clusters in social networks. *Internet Mathematics*, 5(1-2):155–174, 2008.

[107] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC07*, 2007.

[108] Alexander M. Mood. *Introduction to the Theory Of Statistics*, pages 508–511. McGraw-Hill Inc., 1963.

[109] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395.

[110] Patrick Alfred Pierce Moran. A mathematical theory of animal trapping. *Biometrika*, 38(3-4):307–311, 1951.

[111] Movielens. http://movielens.umn.edu.

[112] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[113] J. R. Norris. *Markov Chains.* Cambridge University Press, 1998.

[114] Romualdo Pastor-Satorras, Alexei Vázquez, and Alessandro Vespignani. Dynamical and correlation properties of the internet. *Phys. Rev. Lett.*, 87:258701, Nov 2001.

[115] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:406–413, 7 1955.

[116] C. G. J. Petersen. The yearly immigration of young plaice into the limfjord from the german sea. Report 6, Danish Biological Station, 1895. pp. 5–84.

[117] Amir H. Rasti, Mojtaba Torkjazi, Reza Rejaie, Nick Duffield, Walter Willinger, and Daniel Stutzbach. Evaluating sampling techniques for large dynamic graphs. Technical Report CIS-TR-08-01, Department of Computer and Information Science, University of Oregon, September 2008.

[118] S. Redner. How popular is your paper? an empirical study of the citation distribution. In *European Physical Journal*, B 4, 1998.

[119] G. Siganos S. L. Tauro, C. Palmer and M. Faloutsos. A simple conceptual model for the internet topology. In *Global Telecommunications Conference, volume 3*, GLOBECOM 01, pages 1667–1671, 2001.

[120] Purnamrita Sarkar and Andrew W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In Ronald Parr and Linda C. van der Gaag, editors, *UAI*, pages 335–343. AUAI Press, 2007.

[121] Ajit P. Singh, Asela Gunawardana, Chris Meek, and Arun C. Surendran. Recommendations using absorbing random walks. North East Student Colloquium on Artificial Intelligence (NESCAI), 2007.

[122] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. *Networking, IEEE/ACM Transactions on*, 16(2):267–280, April 2008.

[123] Daniel Stutzbach and Reza Rejaie. On unbiased sampling for unstructured peer-to-peer networks. In *Proc. ACM IMC*, pages 27–40, 2006.

[124] DBLP Team. Dblp xml records. http://dblp.uni-trier.de/xml/, February 2014.

[125] Twitter. http://twitter.com/.

[126] John von Neumann. Various Techniques Used in Connection with Random Digits. *J. Res. Nat. Bur. Stand.*, 12:36–38, 1951.

[127] R. Wagner. Tail estimates for sums of variables sampled by a random walk. *Combinatorics,Probability, and Computing*, 17(2), 2008.

[128] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[129] Thomas Williams and Colin Kelley. Gnuplot. http://www.gnuplot.info/, 2014.

[130] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM.

[131] M. Yamashita. Personal Communication, 2012.

[132] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):pp. 452–473, 1977.

[133] Liyan Zhang, Jie Xu, and Chunping Li. A random-walk based recommendation algorithm considering item categories. *Neurocomputing*, 120(0):391 – 396, 2013.

[134] Liyan Zhang, Kai Zhang, and Chunping Li. A topical pagerank based algorithm for recommender systems. In Sung-Hyon Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat-Seng Chua, and Mun-Kew Leong, editors, *SIGIR*, pages 713–714. ACM, 2008.

[135] Yin Zhu, Erheng Zhong, Sinno Jialin Pan, Xiao Wang, Minzhe Zhou, and Qiang Yang. Predicting user activity level in social networks. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 159–168, New York, NY, USA, 2013. ACM.

[136] Calvin Zippin. The removal method of population estimation. *The Journal of Wildlife Management*, 22(1):pp. 82–90, 1958.

Acronyms

i.i.d. *independent and identically distributed*. 123, 139, 146, 214

uar uniformly at random. 23, 30, 31, 34, 44, 45, 48, 49, 52, 54, 57, 58, 61, 63, 69, 81, 82, 126, 127, 128, 129, 130, 131, 135, 138, 140, 141, 144, 155, 222, 270

whp with high probability. 52, 53, 56, 147, 153, 206, 212, 213, 219, 220, 221, 229

BA model *Barabási-Albert graph generation model*. 17, 45, 46, 47, 52, 54

BFS Breadth-First Search. 27, 89, 90

c.d.f. cumulative distribution function. 55, 139, 140

CFRP Cycle formula of regenerative processes. 19, 20, 21, 31, 143, 149, 150, 154, 160, 162, 163, 169, 171, 179, 186, 192, 198, 199, 205, 271, 272

CS Computer Science. 33

IR ItemRank. 245

K–S test Kolmogorov-Smirnov Test. 90, 138, 139, 140

LRW Lazy Random Walk. 30, 107, 108, 109

MHRW Metropolis Hastings Random Walk. 30, 89, 90, 109, 110, 131, 132, 143, 270

OLSN Online Social Network. 20, 23, 25, 26, 28, 29, 33, 34, 35, 38, 39, 41, 44, 58, 60, 62, 64, 67, 70, 84, 87, 88, 89, 117, 119, 132, 144, 155, 157, 186, 199, 229, 233, 268, 270

p.d.f. probability distribution function. 55

P2P Peer-To-Peer. 70, 89, 211

RDS Respondent-Driven Sampling. 90

RW Random Walk. 17, 27, 28, 29, 30, 31, 32, 39, 70, 71, 72, 89, 91, 92, 98, 101, 110, 111, 115, 121, 127, 142, 143, 144, 145, 153, 155, 163, 164, 206, 211, 237, 238, 266, 269, 270, 271, 272, 273, 274

RWRW Re-Weighted Random Walk. 89, 90, 131, 270

SCC Strongly Connected Component. 38, 39

SRW Simple Random Walk. 18, 19, 30, 31, 70, 71, 89, 90, 103, 104, 105, 106, 108, 109, 110, 113, 115, 131, 141, 143, 146, 147, 153, 154, 155, 156, 166, 169, 179, 198, 199, 205, 222, 254, 263, 266, 270, 272, 273

w.r.t. with respect to. 35, 87, 106, 124, 151

WCC Weakly Connected Component. 38, 39, 179, 199

WRW Weighted Random Walk. 18, 30, 32, 105, 106, 108, 109, 110, 112, 113, 115, 131, 140, 143, 145, 147, 154, 155, 156, 160, 162, 206, 222, 266, 270, 271, 272, 273, 274

WWW World Wide Web. 23, 25, 26, 33, 34, 35, 36, 37, 38, 45, 53, 87, 89, 117, 144, 155, 208, 210, 268, 270