RANDOM SAMPLING AND GENERATION OVER DATA STREAMS AND GRAPHS

XUESONG LU

(B.Com., Fudan University)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

Xuesong Lu

January 9, 2013

Acknowledgements

I would like to thank to my PhD advisor, Professor Stéphane Bressan, for supporting me during the past four years. I could not have finished this thesis without his countless guide and help. Stephane is a very friendly man with profound wisdom and humor full-filled in his brain. It has been a wonderful experience to work with him in the past four years. I was always able to get valuable suggestions from him whenever I encountered problems not only in my research, but also in the everyday life. I am really grateful to him.

I would like to thank to Professor Phan Tuan Quang, who supported me as a research assistant in the past half a year. I would also like to thank to my labmates, Tang Ruiming, Song Yi, Sadegh Nobari, Bao Zhifeng, Quoc Trung Tran, Htoo Htet Aung, Suraj Pathak, Wang Gupping, Hu Junfeng, Gong Bozhao, Zheng Yuxin, Zhou Jingbo, Kang Wei, Zeng Yong, Wang Zhenkui, Li Lu, Li Hao, Wang Fangda, Zeng Zhong, as well as all the other people with whom I have been working together in the past four years. I would also like to thank to my roommates, Cheng Yuan, Deng Fanbo, Hu Yaoyun and Chen Qi, with whom I spent wonderful hours in daily life.

I would like to thank to my parents, who raised me up for twenty years and supported my decision to pursue the PhD degree. You are both the greatest people in my life. I love you, Mum and Dad!

Lastly, I would like to thank to my beloved, Shen Minghui, who accompanied me all the time, especially in those days I got sick, in those days I worked hardly on papers and in those days I traveled lonely to conferences. I am the most fortunate man in the world since I have her in my life.

Contents

Summary

Sampling or random sampling is a ubiquitous tool to circumvent scalability issues arising from the challenge of processing large datasets. The ability to generate representative samples of smaller size is useful not only to circumvent scalability issues but also, per se, for statistical analysis, data processing and other data mining tasks. Generation is a related problem that aims to randomly generate elements among all the candidate ones with some particular characteristics. Classic examples are the various kinds of graph models.

In this thesis, we focus on random sampling and generation problems over data streams and large graphs. We first conceptually indicate the relation between random sampling and generation. We also introduce the conception of three relevant problems, namely, construction, enumeration and counting. We reveal the malpractice of these three methods in finding representative samples of large datasets. We propose problems encountered in the processing of data streams and large graphs, and devise novel and practical algorithms to solve these problems.

We first study the problem of sampling from a data stream with a sliding window. We consider a sample of fixed size. With the moving of the window, the expired data have null probability to be sampled and the data inside the window are sampled uniformly at random. We propose the First_In_First_Out (FIFO) sampling algorithm. Experiment results show that FIFO can maintain a nearly random sample of the sliding window with very limited memory usage.

Secondly, we study the problem of sampling connected induced subgraphs of fixed size uniformly at random from original graphs. We present four algorithms that leverage different techniques: Rejection Sampling, Random Walk and Markov Chain Monte Carlo. Our main contribution is the Neighbour Reservoir Sampling (NRS) algorithm. Compared with other proposed algorithms, NRS successfully realize the compromise between effectiveness and efficiency.

Thirdly, we study the problem of incremental sampling from dynamic graphs. Given an old original graph and an old sample graph, our objective is to incrementally sample an updated sample graph from the updated original graph based on the old sample graph. We propose two algorithms that incrementally apply the Metropolis algorithm. We show that our algorithms realize the compromise between effectiveness and efficiency of the state-of-the-art algorithms.

Fourth, we study the problem of generating random graphic sequences. Our target is to generate graphic sequences uniformly at random from all the possible graphic sequences. We propose two sub-problems. One is to generate random graphic sequences with prescribed length. The other is to generate random graphic sequences with prescribed length and sum. Our contribution is the original design of the Markov chain and the empirical evaluation of mixing time.

Lastly, we study the fast generation of Erdős-Rényi random graphs. We propose an algorithm that utilizes the idea of pre-computation to speedup the baseline algorithm. Further improvements can be achieved by paralleling the proposed algorithm.

Overall, the main difficulty revealed in our study is how to devise effective algorithms that generate representative samples with respect to desired properties. We shall, analytically and empirically, show the effectiveness and efficiency of the proposed algorithms.

List of Tables

List of Figures

Chapter 1

Introduction

A recurrent challenge for modern applications is the processing of large datasets [13, 28, 106]. Two kinds of large datasets are constantly encountered in contemporary applications. They are data streams and large graphs.

A data stream is a ordered sequence **D** of continuous data d_i, each of which arrives in a high speed and usually can be processed only once. Examples of data streams include telephone records, stock quotes, sensor data, Internet traffic, etc. Typically, data streams contain too large amount of data to fit in main memory due to its continuity and high arrival rate. For example, [7] reports that during the first half of 2011, Twitter users sent 200 million Tweets per day. These tweets thereby make up a high-speed, continuous and endless information stream.

A graph is an abstract representation of a set of vertices V where some pairs of vertices are connected by a set of edges E. For example, in an email network, the senders and the receivers are the vertices, and there is an edge connecting a sender and a receiver if they send emails to each other. Modern real life graphs usually consist of at least millions of vertices and billions of edges. For instance, the social graph of Facebook was reported to have about 721 million active users and

68.7 billion friendship edges by May 2011 [107]. Therefore it is often impossible to directly apply graph analysis algorithms on real graphs because of the high complexity of the algorithms.

1.1 Random Sampling and Generation

One way to circumvent scalability issues arising from the above challenge is to replace the processing of very large datasets by the processing of representative samples of manageable size. This is *sampling* or *random sampling.* The ability to generate representative samples of smaller size is useful not only to circumvent scalability issues but also, per se, for statistical analysis, data processing and other data mining tasks. Examples include diverse applications such as data mining [59, 95, 106, 115], query processing [13, 25], graph pattern mining [50, 51], sensor data management [28, 99], etc.

Over time, a series of sampling algorithms have been proposed to cater for different problems. In 1980s, *reservoir sampling* is first introduced by McLeod et al. in [82] and revisited by Vitter in [112]. The algorithm uniformly at random selects a sample with fixed size from a data stream with unknown length. Later *random pairing* and *resizing samples* [37] are proposed based on reservoir sampling to cater for the problem when there are deletions in the original data stream. Despite the extensive discussion of the uniform sampling, modern applications show their preference to recent data. Aggarwal [8] proposes an algorithm to give bias to recent data in the stream. The algorithm samples data with probability exponentially proportional to its arrival time. The later a data arrivals, the higher probability it is sampled. On the other hand, Babcock et al. [14] consider another kind of biased sampling. Rather than sampling from the entire history of the stream, they investigate how a sample can be continuously and uniformly generated within a sliding window containing only the recent data. In addition to sampling data streams, the problem of sampling from large graphs arises these years. The general purpose of graph sampling is to sample representative subgraphs that preserve desired properties of the original graphs. In the pioneering paper, Leskovec et al. [70] discuss several possible sampling techniques for graph sampling. They evaluate the discussed algorithms on their abilities of preserving a list of selected properties of original graphs. Then Hübler et al. [56] propose Metropolis algorithms to improve the sample quality. Later Maiya et al. [80] propose algorithms to sample the community structure in large networks. Other sampling problems include time-based sampling [36, 104, 9], snowball sampling [20, 114] and so on.

Another problem that is related to sampling is *generation*. A generation problem is defined as randomly generating one or more solutions among all the possible ones with some particular characteristics. This is the case when one discusses the graph models. For instance, the classic random graph model, or Erdős-Rényi model, proposed by Gilbert [40] and Erdős et al. [33], randomly generates graphs with given number of vertices and the probability to link each pair of the vertices, or with given number of vertices and edges. Then a successive of graph models are proposed to simulate the real graphs, including the Watts and Strogatz model [113], the Barabási-Albert model [15], the Forest Fire model [71], etc. All these graph models are randomly generating graphs with desired properties of the real graphs, among all the possible ones. Some other literatures discuss the problem of generating random graphs with prescribed degree sequences [84, 111, 38]. This is also the case when one discusses random generation of synthetic databases. Examples include fast generation of large synthetic database [45], generation of spatio-temporal datasets [105], data generation with constraints [12], etc.

As discussed above, sampling is extracting representative samples from the original population. Indeed, the sampling process is equivalent to randomly generating representative samples among all the possible samples. For example, given n vertices and m edges of a graph the Erdős-Rényi model randomly selects m edges from $\frac{n(n-1)}{2}$ possible edges. This is sampling. On the other hand, the Erdős-Rényi model is randomly generating graphs among all the graphs with n vertices and m edges. This is generation. Therefore sampling and generation are equivalent problems interpreted from two different angles.

In this thesis, we study random sampling and generation problems over data streams and graphs. We propose novel algorithms to solve these problems.

1.2 Construction, Enumeration and Counting

Before further discussing sampling and generation, we introduce three related problems. They are construction, enumeration and counting.

A construction problem[1] aims to find an arbitrary element with desired characteristics. Note that a generation problem aims to find a random element. For example, to construct a sample of size n from a data stream, one can simply select the first n data. To construct an induced subgraph with n vertices from an original graph, one can arbitrarily select n vertices and construct the corresponding induced subgraph. There are two classic construction algorithms for extracting subgraphs with given sizes (number of vertices). They are the *Depth First Search* (DFS) algorithm and the *Breadth First Search* (BFS) algorithm [65]. DFS starts at some vertex and explores as far as possible along each branch before backtracking, until desired number of vertices are selected. BFS starts at some vertex and explores all the neighbours of the vertex. For each neighbour, BFS then recursively explores

[1]We present the results of a construction problem in Appendix C

its unvisited neighbours, until desired number of vertices are selected. Both of the algorithms construct deterministic subgraphs once the first vertex is selected. Another example is the *simple* algorithm [14]. The algorithm is proposed for sampling from a data stream with a sliding window. The data in the window are supposed to be sampled uniformly at random. The algorithm samples the first window using the standard reservoir sampling. However, the consecutive samples are produced by construction. Whenever a data in the current sample is expired, the newly arrival data is inserted into the sample. This algorithm reproduces periodically the same sample design once the sample of the first window is generated. As what we see, the element constructed by a construction method is also a sample of the underlying population. The problem with construction is that it usually produces unrepresentative samples because the sample design is usually deterministic. Instead, random sampling and generation methods are an option when construction methods cannot produce representative samples.

One naive method to solve a generation problem is to enumerate all the elements and randomly select some of them. The former processing is called *enumeration*. For example, Harary et al. [49] discuss in their book the enumeration of graphs and related structural configurations. Another classic enumeration problem is the *maximal clique enumeration* problem [11]. A clique is a complete subgraph. A maximal clique is a clique that is not contained in any other cliques. The maximal clique enumeration problem aims to enumerate all maximal cliques in a graph. The problem is *NP*-hard. Generation by enumeration is often impractical when space of elements is large. In such a scenario, practical sampling and generation algorithms are required.

Another relevant problem is *counting*. The problem aims to count the number of possible elements with desired characteristics without enumerating. If the number of elements can be efficiently counted, it is possible to assign any probability distribution on the elements and generate random elements according to the distribution. For example, the problem *"how many different samples of size n are there given a population of size m?"* can be easily solved using the combination formula $\binom{m}{n}$. One can assign each sample probability $1/\binom{m}{n}$ and generate the samples uniformly at random. However, many counting problems are difficult as there is no direct approach to calculate the corresponding numbers. For example, there is no known formula to solve the problem *"how many distinct graphic sequences with length n are there?"*. One could obtain the result by enumerating all the distinct graphic sequences with length n. However, it is impractical. In fact, there are a category of counting problems called $\sharp P$ problems which are associated with the decision problems in the set NP. For example the problem *"Are there any subsets of a list of integers that add up to zero?"* is an NP problem while the problem *"How many subsets of a list of integers add up to zero?"* is a $\sharp P$ problem. If a problem is in $\sharp P$ and every $\sharp P$ problem can be reduced to it by polynomial-time counting reduction, the problem is in $\sharp P$-complete. Famous examples include *"How many different variable assignments will satisfy a given DNF formula?"*, *"How many perfect matchings are there for a given bipartite graph?"*, etc. More examples of $\sharp P$-complete problems can be found in Appendix A. If a counting problem is in $\sharp P$-complete, it is impractical to sample via counting. Instead, we are interested in designing efficient sampling and generation algorithms.

1.3 Contributions

In this thesis, our main contribution is the novel design of sampling and generation algorithms for different problems over data streams and graphs. We list the research gaps and the achievements so far as follows.

1.3.1 Sampling from a Data Stream with a Sliding Window

Sampling streams of continuous data with limited memory, or reservoir sampling, is a utility algorithm. Standard reservoir sampling maintains a random sample of the entire stream as it has arrived so far. This restriction does not meet the requirement of many applications that need to give preference to recent data. Babcock et al. discuss the problem of sampling from a sliding window in [14]. They propose the simple algorithm and the chain-sample algorithm. However, the two algorithms suffer from different drawbacks, respectively. The simple algorithm produces periodical sample design, and the chain-sample algorithm requires high memory usage. Moreover, it is unclear how to sample more than one elements with the chain-sample algorithm.

We propose an effective algorithm, which is very simple and therefore efficient, for maintaining a near random fixed size sample of a sliding window [79]. Indeed our algorithm maintains a biased sample that may contain expired data. Yet it is a good approximation of a random sample with expired data being present with low probability. We analytically explain why and under which parameter settings the algorithm is effective. We empirically evaluate its performance and compare it with the performance of existing representatives of random sampling over sliding windows and biased sampling algorithm.

1.3.2 Sampling Connected Induced Subgraphs Uniformly at Random

A recurrent challenge for modern applications is the processing of large graphs. Given that the graph analysis algorithms are usually of high complexity, replacing the processing of original graphs by the processing of representative subgraphs of smaller size is useful to circumvent scalability issues. For such purposes adequate graph sampling techniques must be devised. Despite the fact that many graph sampling problems have been proposed in the past few years, little work has been done on sampling connected induced subgraphs. In fact, connected induced subgraphs naturally preserve local properties of original graphs.

We study the uniform random sampling of a connected subgraph from a graph [75]. We require that the sample contains a prescribed number of vertices. The sampled graph is the corresponding induced graph. We devise, present and discuss several algorithms that leverage three different techniques: Rejection Sampling, Random Walk and Markov Chain Monte Carlo. We empirically evaluate and compare the performance of the algorithms. We show that they are effective and efficient but that there is a trade-of, which depends on the density of the graphs and the sample size. We propose one novel algorithm, which we call Neighbour Reservoir Sampling, that very successfully realizes the trade-of between effectiveness and efficiency.

1.3.3 Sampling from Dynamic Graphs

The graphs encountered in modern applications are dynamic: edges and vertices are added or removed. However, existing graph sampling algorithms are not incremental. They were designed for static graphs. If the original graph changes, the sample graph must be entirely recomputed.

We present incremental graph sampling algorithms preserving selected properties, by applying the Metropolis algorithms [76]. The rationale of the proposed algorithms is to replace a fraction of vertices in the old sample with newly updated vertices. We analytically and empirically evaluate the performance of the proposed algorithms. We compare the performance of the proposed algorithms with that of baseline algorithms. The experiment results on both synthetic and real graphs show that our proposed algorithms realize a compromise between effectiveness and efficiency, and, therefore provide practical solutions to the problem of incrementally sampling from the large dynamic graphs.

1.3.4 Generating Random Graphic Sequences

The graphs that arise from concrete applications seem to correspond to models with prescribed degree sequences. A lot of work has discussed the problem of generating random graphs with prescribed degree sequences [84, 111, 38]. They randomly produce graphs with particular characteristics with the given prescribed degree sequences. An underlying problem of this graph generation problem is therefore the generation of degree sequences.

We present four algorithms for the random generation of graphic sequences [74]. We have proved their correctness. We empirically evaluate their performance. Two of these algorithms that generate random graphic sequences according to the underlying distribution of random graphs are trivial and are as effective and efficient as the corresponding graph generation algorithms. The two other algorithms generate graphic sequences uniformly at random. To our knowledge these algorithms are the first non-trivial algorithms proposed for this task. The algorithms that we propose are Markov chain Monte Carlo algorithms. Our contribution is the original design of the Markov chain and the empirical evaluation of mixing time.

1.3.5 Fast Generation of Random Graphs

Today, several database applications call for the generation of random graphs. A fundamental, versatile random graph model adopted for that purpose is the Erdős-Rényi $\Gamma_{v,p}$ model. This model can be used for directed, undirected, and multipartite graphs, with and without self-loops; it induces algorithms for both graph generation and sampling, hence is useful not only in applications necessitating the generation of random structures but also for simulation, sampling and in randomized algorithms. However, the commonly advocated algorithm for random graph generation under this model performs poorly when generating large graphs.

We propose PreZER, an alternative algorithm with certain pre-computation for random graph generation under the Erdős-Rényi model [90]. Our extensive experimental study shows significant speedup for PreZER with respect to the baseline algorithm.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Section 2 introduces background knowledge and takes a detailed review of related work. Section 3-7 present the main contributions in our work, which includes sampling of a data stream with a sliding window (Section 3), sampling connected induced subgraphs uniformly at random (Section 4), sampling dynamic graphs (Section 5), generating random graphic sequences (Section 6) and fast generation of random graphs (Section 7). Section 8 proposes the problems and possible directions in the future work. Finally, we conclude in Section 9. We also show the results of some relevant work in Appendix.

Chapter 2

Background and Related Work

2.1 Markov Chain Monte Carlo

Many literatures [56, 80, 43, 84] devise algorithms that belong to the *Markov Chain Monte Carlo (MCMC)* method [52]. Markov chain Monte Carlo algorithms are random walks on Markov chains. They stem from the Monte Carlo methods used in applied statistics where they were used for simulation and sampling [52]. Sinclair, in his monograph [100], has formalized and popularized their use for random generation (sampling) and counting.

An MCMC algorithm builds and randomly walks on a Markov chain whose states correspond to the objects being sampled. The current state depends only on its adjacent states. It is not necessary that all the states of the Markov chain correspond to object being sampled as a rejection mechanism can filter out those undesirable objects. Nevertheless, if the Markov chain is carefully designed, if it is finite and ergodic (irreducible and aperiodic), then it has a stationary distribution of random walk. Namely, the stationary probability vector $\boldsymbol{\pi}_n$ of the Markov chain satisfies $\boldsymbol{\pi}_n = \boldsymbol{\pi}_n \mathbf{P}_{n\times n}$, where $\mathbf{P}_{n\times n}$ is the transition matrix of the Markov chain, specifying the transition probability between every pair of states. For instance, the stationary distribution is uniform if the graph of the underlying Markov chain is regular and if the transition probability from s to $s\prime$ is defined as follows:

$$p(s, s\prime) = \begin{cases} \frac{1}{d_s} & \text{if} \quad s \text{ and } s\prime \text{ are adjacent} \\ 0 & \text{else,} \end{cases}$$

where d_s is the degree of state s in the graph of the Markov chain.

The mixing time of a Markov chain is the minimum number of random walk steps t required to reach the stationary distribution. A sufficiently long random walk, longer than the mixing time of a Markov chain, will reach states at random according approximately to the stationary distribution. Formally, following [100], let $MC(x)$ be a family of ergodic Markov chains parameterized on strings $x \in \Omega$. For each x, let $\Delta^{(x)}(t)$ denote the distance[1] between the stationary distribution and the distribution of $MC(x)$ from any initial state after t steps, the mixing time $T^x(\varepsilon)$ with error ε is defined as follows.

$$T^{(x)}(\varepsilon) = min\{t \in N : \Delta^{(x)}(t') \leq \varepsilon \quad \text{for all} \quad t' \geq t, 0 < \varepsilon \leq 1\}.$$

Such a family is said to be rapidly mixing if and only if there exists a polynomial bounded function $q : N \times R^+ \to N$ such that

$$T^{(x)}(\varepsilon) \leq q(|x|, \log \varepsilon^{-1})$$

For a given generation problem the challenge is to devise a rapidly mixing ergodic Markov chain whose states are the objects to be generated with the desired stationary distribution. Generally, there are two methods to obtain the desired

[1]There are several different definitions of distance such as the total variation distance.

stationary distribution. One is using the Metropolis algorithm [83]. This method revises the stationary distributions by controlling the transition probability between two states, for instance, according to their degrees in the graph of the Markov Chain. The other method is modifying the weights of certain edges in the Markov Chain, so that the sum of weights incident to each state is proportional to the desired stationary distribution [100].

The Metropolis algorithm [83] can draw samples from an ergodic Markov chain with any desired probability distribution $\boldsymbol{\pi}$ by satisfying the *detailed balance* condition,

$$\pi(s)p(s, s\prime) = \pi(s\prime)p(s\prime, s), \tag{2.1}$$

where $\pi(s)$ is the stationary probability of state s and $p(s, s\prime)$ is the transition probability from s to $s\prime$, for any state s and $s\prime$. The transition probability $p(s, s\prime) = t(s, s\prime) \times a(s, s\prime)$, where $t(s, s\prime)$ is the probability of the transition from s to $s\prime$ and $a(s, s\prime)$ is the probability of accepting the transition. To ensure the detailed balance, the acceptance probability has to be set as $a(s, s\prime) = \min(1, \frac{\pi(s\prime)}{\pi(s)} \times \frac{t(s,s\prime)}{t(s\prime,s)})$. If the constructed Markov chain is regular, that is, all the states have the same number of adjacent states, we have $t(s, s\prime) = t(s\prime, s)$. In such kind of scenario, the acceptance probability is simplified as $a(s, s\prime) = \min(1, \frac{\pi(s\prime)}{\pi(s)})$. It is easy to prove that the detailed balance holds for the Markov chain if only it holds for any two adjacent states of the Markov chain. We prove this property in Chapter 5.

The other method to draw samples with any desired probability distribution, suggested by Sinclair in [100], is to modify the weights of edges corresponding to the transitions between adjacent states of the Markov chain. Namely, the stationary probability of each state is proportional to the sum of the weights of its incident edges in the graph of the underlying Markov chain, with the following transition matrix **P**,

$$p(s, s\prime) = \begin{cases} \frac{w_(s,s\prime)}{\sum_{l \in adj(s)} w_(s,l)} & \text{if} \quad s \text{ and } s\prime \text{ are adjacent} \\ 0 & \text{else,} \end{cases}$$

where $w_(s, s\prime)$ is the weight of edge corresponding to the transition from state s to $s\prime$.

2.2 Sampling a Stream of Continuous Data

For the problem of sampling a stream of continuous data, previous work focuses mainly on two areas. One is sampling the entire history uniformly at random. The other is giving preference to recent data. The former area includes the work such as *Reservoir Sampling* [82, 112], *Random Pairing* and *Resizing Samples* [37]. The latter includes the work such as *Biased Reservoir Sampling* [8], the *Simple* algorithm, *Chain-Sample* [14] and *Partition Sampling* [22].

McLeod et al. [82] first introduce the Reservoir Sampling algorithm. Given a data stream with unknown length and a sample size n, the algorithm inserts directly the first n data into the sample. Then for each of the subsequent data, the algorithm inserts it into the sample with probability n/t, where t is the length of the stream after this insertion. If the newly arrival data is inserted, the algorithm discard one data in the current sample uniformly at random. Reservoir Sampling always maintains a uniform random sample of the entire data stream. Vitter [112] revisits the Reservoir Sampling algorithm and proposes the optimized version algorithm Z. The basic idea is to skip data that are not going to be selected according to the selection probability, and rather select the index of next data. A formula is derived to compute the number of data that are skipped over before the next data is chosen for the reservoir. This technique reduces the number of data that need to be processed and thus the number of calls to RANDOM, which is a function to generate a uniform random variable between 0 and 1. With the increasing of the length of the data stream, more and more data are skipped directly.

The basic Reservoir Sampling algorithm handles only the insertion of data of the original stream. Gemulla et al. [37] take a dip into the Reservoir Sampling by considering both insertions and deletions of a data stream. They propose the Random Pairing and Resizing Samples algorithms. The basic idea behind Random Pairing is to avoid accessing the original data stream by considering each new insertion as a compensation for the previous deletion. In the long term, every deletion from the data stream is eventually compensated by a corresponding insertion. The algorithm maintains two counters c_1 and c_2, which denote the numbers of uncompensated deletions in the sample S and in the original data stream R, respectively. Initially c_1 and c_2 are both set to 0. If $c_1+c_2=0$, the Reservoir Sampling algorithm is applied. If $c_1+c_2 \neq 0$, the new data has probability $c_1/(c_1+c_2)$ to be selected into S; otherwise, it is discarded. Then c_1 or c_2 are modified accordingly. Random Pairing is proposed for the scenario that the size of the original data stream is relatively stable. On the other hand, they consider the data stream that is growing in the long run, so that the upper bound of sample size grows with the length of the data stream. The propose the Resizing Samples algorithm. The general idea is to generate a sample S of size at most n from the initial data stream R and, after some finite transactions of insertions and deletions, produce a sample S' of size n' from the new base data stream R', where $n < n' < |R|$.

Modern applications show their preference to recent data. To cater for this problem, Aggarwal [8] proposes the Biased Reservoir Sampling algorithm that samples each data with probability proportional to its arrival time. The later a data is present in the stream, the higher probability it is sampled with. In particular, the probability of the r^{th} data included in the sample at the arrival of the t^{th} data is proportional to a bias function $f(r,t) = e^{-\lambda(t-r)}$, where λ is the bias rate lying between 0 and 1. They define the bias using this exponential function because the probability distribution can be easily maintained by modifying Reservoir Sampling, as they proved in [8]. The algorithm first maintains an empty sample of capacity $n = [1/\lambda]$. Assume that at the arrival of the t^{th} data, the fraction of the sample filled is $F(t)$. The $(t+1)^{th}$ data is deterministically inserted into the sample. However, the deletion of an old data in the sample is not necessary. The algorithm flips a coin with success probability $F(t)$. In case of a success, a randomly selected data in the old sample is discarded. Otherwise, no deletion occurs and the sample size increases by 1. Notice that once the sample is fully filled, each newly arrival data replaces one data randomly selected from the old sample with probability 1.

Another kind of biased sampling is sampling with a sliding (moving) window. Babcock et al. first consider this problem in [14]. Given a data stream with unknown length, a sample size n and a window size w, the sampling scheme should always maintain a uniform random sample of the most recent w data of the stream. Notice that previous work maintains a sample of the entire data stream, whereas sliding window sampling maintains a sample of recent data only. They propose the Simple algorithm and the Chain-Sample algorithm. The Simple algorithm first generates a sample of size n from the first w data using the reservoir sampling algorithm. Then the window moves on the stream. The current sample is maintained until a newly arrival data causes an old data in the sample to be expired (outside the window). The new data is then inserted directly into the sample and the expired data is directly discarded. This algorithm can efficiently maintain a uniform random sample of the sliding window. However, the sample design is re-produced for every tumbling window. If the i^{th} data is in the sample of the current window, the $(i + cw)^{th}$ data is guaranteed to be included into the sample of some window in future, where c is an arbitrary integer constant. To avoid this problem, they propose the Chain-Sample algorithm. When the i^{th} data enters the window, it is selected to be the sample with probability $\frac{Min(i,w)}{w}$. If the data is selected, the index of the data that replaces it when it expires is uniformly chosen from $i + 1$ to $i+w$. When the data with the selected index arrives, the algorithm puts it into the sample and calculates the new replacement index, etc. Thus, a chain of elements that can replace the outdated data is built. Chain-Sample generates a sample of size 1 for each chain. In order to obtain a sample of size n, n chains need to be maintained. However, the Chain-Sample algorithm requires high memory usage, as analyzed in [14].

A recent work proposes the Partition Sampling algorithm for sliding window sampling on a data stream with $O(n)$ memory usage [22]. The basic idea is to partition the data stream into disjoint buckets $B(iw, (i + 1)w)$, $i = 0, 1, \dots,$ where w is the window size, and draw a sample from the most recent two buckets based on some principle. At any time, there are one active bucket and at most one partial bucket. A bucket is considered as "active" if not all the data of the bucket have been expired. A bucket is considered as "partial" if not all the data of the bucket have arrived yet. The algorithm is then as follows. Denote by U the active bucket and by S_U the sample of size n drawn from U using the Reservoir Sampling algorithm. If there is no partial bucket, the final sample S is simply equal to S_U. Otherwise, denote by V the partial bucket. If all the data of S_U are not expired, $S = S_U$. Otherwise, let U_e be the set of data of U that are expired, U_a be the set of data of U that are not expired (still active), and V_a be the set of data of V that have arrived. Let i be the number of expired data in S_U, that is, $i = |U_e \cap S_U|$.

The algorithm draws a sample S_V^i of size i of V_a from S_V. Finally, the sample S is equal to $(S_U \cap U_a) \cup S_V^i$. They prove the correctness of the algorithm. Since there are at mot two buckets under processing at any time and the Reservoir Sampling algorithm requires $O(n)$ memory, the total memory of the algorithm is $O(n)$.

2.3 Graph Sampling

For the problem of graph sampling, existing work focuses mainly on two different sampling purposes. One purpose is to generate subgraphs of smaller sizes that preserve selected properties of the original graph, such as degree distribution, component distribution, average clustering coefficient, community structure, etc. Although these algorithms have a random component, they are primarily construction algorithms and are not designed with the main concern of randomness and uniformity of the sampling. In general the distribution from which these random graphs are sampled are not known. Representatives of literatures include sampling from large graphs [70], Metropolis Graph Sampling [56], and sampling community structure [80]. The other purpose is to sample subgraphs of interest uniformly at random from the original graphs. Representatives of literatures include sampling random URL [54], sampling network motifs [61] and sampling unbiased Facebook users [43].

In the pioneering paper [70], Leskovec et al. discuss ten candidate sampling algorithms aiming at preserving a selected list of graph properties. They also propose two sampling goals, namely, scale-down sampling and back-in-time sampling. The former aims to create a sample graph S that is similar to the original graph G. The latter aims to find a sample graph S that is similar to the original graph G when it has the same size as S. The algorithms discussed include sampling by random node selection, sampling by random edge selection and sampling by exploration of the graph. They consider the graph properties as distribution, including degree distribution, clustering coefficient distribution, the distribution of component size, hop-plot, etc. They empirically and comparatively evaluate the candidate algorithms by measuring the similarity between the generated sample graphs and the original graphs on nine static properties and five evolving properties. Among the discussed algorithms, Random Walk Sampling and Forest Fire Sampling have the best overall performance. The Random Walk Sampling algorithm selects uniformly at random a starting vertex and simulates a random walk on the graph. In each step, the algorithm jumps back to the starting vertex with a certain probability and restarts the random walk. The algorithm has the problem of getting stuck. The solution is to select a new starting vertex and repeat the above procedure if no enough number of vertices are sampled after a sufficiently long time. Similarly to Random Walk Sampling, Forest Fire Sampling selects a starting vertex v uniformly at random from the graph. Then a random number x is drawn from a geometric distribution with mean $\frac{p_f}{1-p_f}$, where p_f is called forward burning probability. The algorithm hereafter selects x neighbor vertices of v that have not yet been selected. Recursively, the algorithm applies the same process to these newly selected vertices, until enough number of vertices are sampled. If the algorithm gets stuck at some vertex, it selects a new starting vertex uniformly at random and restarts the above procedure.

The algorithms proposed in [70] are practical with respect to preserving different graph properties. However, they do not guarantee to find the most representative sample graphs. To improve the sample quality, Hübler et al. propose the Metropolis Graph Sampling algorithm in [56]. The algorithm generates sample graphs with higher quality and smaller size, compared with the algorithms in [70]. The basic idea is to generate the sample graphs with probability proportional to their qualities using the Metropolis algorithm, and modify the algorithm into an optimization variant by storing the sample graph with the best quality. In particular, the algorithm begins with a subgraph S of size n selected uniformly at random from the original graph G. At each random walk step, one vertex outside S is selected uniformly at random to replace one vertex inside S. The replaced vertex is selected uniformly at random from S. Let S' be the new subgraph. Then they compute the values of $\Delta_{G,\sigma}S$ and $\Delta_{G,\sigma}S'$, where $\Delta_{G,\sigma}S$ is a distance measure of the difference between S and G on the graph property σ. With a success probability $(\frac{\Delta_{G,\sigma}S}{\Delta_{G,\sigma}S'})^p$, S moves to S'; otherwise, the algorithm stays at S. Here p is a parameter that affects the convergence of the constructed Markov chain. In the event of a success, the algorithm stores S as S_{best} if $\Delta_{G,\sigma}S < \Delta_{G,\sigma}S_{best}$. The random walk stops after a sufficient large number of steps, and the algorithm outputs S_{best} as the sample graph. For a certain graph property, the transition probability makes S definitely move to subgraphs more similar to G, and move to subgraphs less similar to G with smaller probability.

In addition to sampling general graph properties, Maiya et al. focus on preserving the community structure of the original graph in [80]. The generated sample graphs can be viewed as stratified samples in that they consist of members from most or all communities in the original graph. They define a conception of expansion factor, or expansion for short, of a subgraph S. Their method is then based on the rationale that better expansion equates to better community representativeness. They propose the Snowball Sampling algorithm and the Markov chain Monte Carlo algorithm. The Snowball Sampling algorithm begins with a subgraph S containing only one vertex which is selected uniformly at random from the original graph G. At each iteration, the algorithm adds a vertex v to S chosen from the adjacent vertices of S, such that v contributes the most to the expansion of S. The iteration terminates when the desired number of vertices are sampled. The Markov Chain Monte Carlo algorithm reproduces the algorithms in [56]. The difference is that the quality of subgraphs is measured using the expansion factor.

The other category of graph sampling is concerned with randomness and uniformity of the samples. The random or uniform sampling is useful for statistical analysis, data mining and simulation, as it naturally and statistically maintains the properties of interest. For instance, Kashtan et al. uses the sampling method to estimate network motifs concentration in [61]. Network motifs are connected subgraphs matching a prescribed pattern (and therefore a prescribed size), which usually have only 3, 4 or 5 vertices. They propose a random sampling algorithm that is proved to be biased. The RVE algorithm that we discuss in Section 4.2.2 is a variant of their work. We will present this algorithm below. However, they devise an estimation method to adjust the bias and calculate the approximate concentrations (frequencies) of sampled motifs. This method is only practical for motifs of small size, due to the high complexity of the computation.

Another example of uniform graph sampling is the problem of obtaining a representative (unbiased) sample of Facebook users studied by Gjoka et al. in [43]. Facebook users form such a graph that each vertex represents a user and an edge connects a pair of vertices if the two corresponding users are friends. The authors do not guarantee to generate a subgraph of fixed size. Instead, they are concerned only with the unbiasedness of the sample. Among the algorithms proposed by them, they find the Metropolis-Hastings Random Walk algorithm and the Re-Weighted Random Walk perform well. The Metropolis-Hastings Random Walk algorithm begins with an arbitrary vertex. At each iteration, the algorithm selects a vertex w uniformly at random from neighbours of the current vertex v. Then v moves to w with probability k_v/k_w, or stay at v otherwise, where k_v is the degree of v. The algorithm always accepts the move towards smaller degree vertices, and rejects some of the moves towards higher degree vertices. This adjusts the bias towards high degree vertices in the original random walk algorithm. The iteration terminates when an unbiased sample of users is obtained. This termination criterion is tested using diagnostic tools [39, 35]. The Re-Weighting Random Walk algorithm conducts a standard random walk on the graph, but corrects the degree bias by an appropriate re-weighting of the measured values using the Hansen-Hurwitz estimator [48]. Re-Weighting Random Walk is rather an estimation of the unbiasedness of the sample than an algorithm that generates an unbiased sample.

2.4 Graph Generation

For the problem of graph generation, a lot of work has been done on generating graphs of a given model or of a prescribed constraint. Random graph generation under specific models and with prescribed constraints can be seen as sampling from a virtual specific sample space consisting of all the possible random graphs. Known graph generation models include the Erdős-Rényi model [40, 33], the Watts and Strogatz (small-world) model [113], the Barabási-Albert (preferential-attachment) model [15] and the Forest Fire model [71]. An interesting constraint is the prescribed degree sequence. Generating graphs uniformly at random with prescribed degree sequence is discussed in [84, 111].

Gilbert [40] and Erdős et al. [33] discuss two algorithms for random graph generation. The two corresponding models together are referred to as the Erdős-Rényi model. The basic model generates a random undirected graph with n vertices by linking each pair of vertices with probability p. Recently, Nobari et al. discuss fast generation of this model using GPU in [90]. An alternative algorithm of the Erdős-Rényi model is to randomly select m edges from all the $\frac{n(n-1)}{2}$ (undirected graphs without self-loop) edges. The model is widely used as a baseline for testing graph analysis algorithm, modeling and simulation. However, the model does not capture the properties in real graphs such as power-law degree distribution and high clustering coefficient.

The classic random graph model (Erdős-Rényi model) generates random graphs such that each pair of vertices is linked with identical probability. The graphs generated in this way have homogeneous structure for vertices while in real life graphs usually exhibit hierarchical organization, where vertices could be partitioned to groups and further to subgroups, etc. Clauset et al. propose a variation of Erdős-Rényi model to generate random graphs with respect to prescribed hierarchical structure in [27]. In particular, for an observed graph G they generate corresponding dendrogram D with probability proportional to the likelihood L that how D is fitting the structural of G, and then re-sample random graphs from D which have similar hierarchical structures with G. This model is similar to Erdős-Rényi model except that the probabilities linking pairs of vertices are inhomogeneous. In this way, the model can generate random graphs with hierarchical structures.

Watts et al. propose the Watts and Strogatz model in [113]. Given the number of vertices n, the average degree k and a rewiring probability β, the algorithm generates an undirected graph in the following way. It arranges the n vertices on a ring and links each vertex with its k neighbours, $k/2$ on each side. Then for each vertex v_i, it takes every edge (v_i, v_j) with $i < j$ and rewires v_i with probability β to v_k selected uniformly at random from remaining vertices. The rewiring avoids to generate loops or multiple edges. This model captures the small-world phenomenon (high clustering coefficient) in real graphs, but fails to reproduces the power-law degree distribution.

Later Barabási et al. propose the Barabási-Albert model in [15]. In this model, the vertices are added to the current graph one by one. Each newly added vertex links to m existing vertices. The m vertices are selected with probability proportional to their degrees. Therefore the old vertices attract the newly added vertices in a rich-get-richer (preferential attachment) way. Graphs generated by this model exhibit power-law degree distribution. Later Klemm et al. propose two extended models. The first model [63] leverages the behaviors of the Watts and Strogatz model and the Barabási-Albert model. The second model [64] generates highly clustered scale-free (power-law degree distribution) graphs.

For the sake of fast generation, Batagelj et al. discuss efficient algorithms for generating graphs of most of the above models in [18].

Recently, Leskovec et al. [71] study the evolution of real graphs. They find that other than the static properties of a graph snapshot such as power-law degree distribution and high clustering coefficient, the real graphs exhibit densification power law and shrinking effective diameters. They propose the Forest Fire model to reproduce these evolving properties. Given a forward burning probability p and a backward burning ratio r, Forest Fire generates a directed graph in the following way. Each newly added vertex v links to an existing vertex w selected uniformly at random from the current graph G. Then v selects x out-links and y in-links of w incident to unvisited vertices. x and y are two random numbers geometrically distributed with mean $p(1-p)$ and $rp/(1-rp)$, respectively. Denote by $w_1, w_2, \ldots, w_{x+y}$ the vertices in the other end of the selected links. v forms out-links to $w_1, w_2, \ldots, w_{x+y}$, and then recursively burns the links of each of these $x+y$ vertices. The model is reported to capture not only the densification power law and a shrinking diameter, but also heavy-tailed in- and out-degrees.

Milo et al. [84] study the problem of uniform generation of random graphs with prescribed degree sequences. They discuss three algorithms. The first is the switching algorithm. It uses a Markov chain to generate a random graph with a given sequence, which is also discussed in [88, 81, 85]. At each iteration, the algorithm selects uniformly at random a pair of edges ($A \to B$, $C \to D$), and exchanges their ends to get edges ($A \to D$, $C \to B$). The iteration terminates after a sufficient large number of steps. The second is the matching algorithm. Initially there is no edge between the vertices. Then the algorithm assigns stubs to the vertices according to the given degree sequence. The edges are created by randomly linking two stubs. The third is the "go with the winners" algorithm. It begins by taking several copies of the initial graph in the matching algorithm. At each iteration, it randomly creates one edge in each graph. If a copy of graph cannot form any more edge at some point, it is dropped. The algorithm periodically cloning each of the surviving graphs to keep enough number of copies. After all the edges are created, the generated graph is selected uniformly at random from all the surviving copies. They compare the three algorithms and find that the switching algorithm and the "go with the winners algorithm" can generate graphs with prescribed degree sequence uniformly at random. The matching algorithm has measurable deviation from the uniform distribution.

Chapter 3

Sampling from a Data Stream with a Sliding Window

3.1 Introduction

The fact that reservoir sampling [82, 112] produces samples of the entire data stream as opposed to samples of data in a sliding window is a clear disadvantage for data stream applications where users need to consider the most recent information. To meet this requirement, Babcock et al. propose the Simple algorithm and the Chain-Sample algorithm in [14]. However, the Simple algorithm reproduces the same sampling design for each tumbling window, and Chain-Sample needs memory that is only statistically bound.

In this work, we consider the problem of maintaining an approximate uniform random sample of a fixed specified size n over a data stream based on a sequence based sliding window model. The algorithm presented in this work belongs to the class of reservoir sampling. It is called FIFO ($\underline{\text{F}}$irst $\underline{\text{I}}$n $\underline{\text{F}}$irst $\underline{\text{O}}$ut) sliding window sampling, or FIFO for short. It maintains a sample of size n and requires a memory of size n. The algorithm is biased towards recent data. The algorithm however does not produce true random samples of the sliding window and may contain expired data. However, as we argue, it can be parameterized to produce almost random samples. It relies on a simple queue data structure. We present some analytical results and empirically and comparatively evaluate its effectiveness. We compare it with the main reservoir and sliding window reservoir algorithms. We conclude that FIFO is both efficient and effective.

3.2 The FIFO Sampling Algorithm

We consider sampling with a sliding window. In detail, we consider a window on a data stream. While the stream evolves, new data comes into the window and expired data goes out, which constitutes a sliding window on the data stream. Our task is to always extract a sample uniformly at random from the current window. Formally, suppose the sample size is n and the window size is w. After the t^{th} data is processed, all the expired data which arrive before the $t - w + 1^{th}$ data should have null probability to be included in the sample while all the subsequent data have probability n/w to be included. The probability distribution is shown as Figure 3.1.

We hence propose a new algorithm to approximately maintain a uniform random sample of a sliding window, which is called First In First Out, or *FIFO* for short, sampling algorithm [79]. The main idea of FIFO is that whenever a new data in the stream arrives, we insert it into the sample with a fixed probability p and simply discard the oldest data in the sample. The complete algorithm is given in Algorithm 1. A critical point is the selection of the inclusion probability p. Below we show that with appropriate selection of the value of p, FIFO approximately

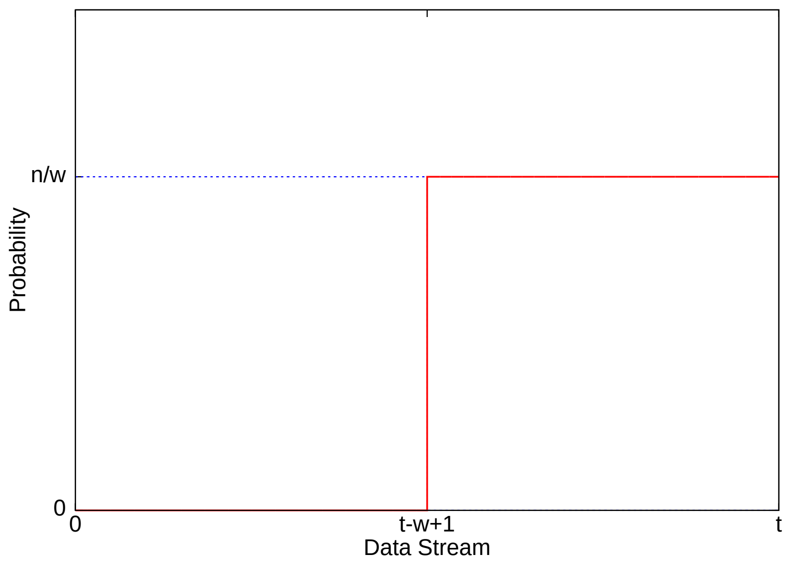


Figure 3.1: The probability distribution of uniform sampling of the window. t is the number of processed data in the stream.

Algorithm 1: FIFO Sampling Algorithm

```
Input: $n$ : sample size, $p$ : inclusion probability, $DS$ : data stream
Output: The sample $S$
1 $S \leftarrow \{\}$;
2 Insert sequentially the first $n$ data of $DS$ into $S$;
3 while NOT $EndOFStream(DS)$ do
4     Randomly generate a number $\varphi$ in the interval $[0, 1)$;
5     if $\varphi < p$ then
6         Insert the next data into $S$;
7         Discard the oldest data in $S$;
8     end
9 end
```

generates a uniform random sample of the sliding window.

3.3 Probability Analysis

In this section, we show the probability distribution of FIFO by giving the probability formula of each data in the stream. The derivation of the formula is based on the facts that, a data is contained in the sample at a certain point if and only if (a) the data is selected into the sample when it is processed and (b) it has not been discarded. Because FIFO always discards the oldest data in the sample, a newly inserted data is not discarded until n of its subsequent data are selected into the sample, that is, if the r^{th} data is inserted into the sample when it is processed and less than n data are inserted into the sample from the $(r+1)^{th}$ data to the t^{th} data, the r^{th} data is still present in the sample after the t^{th} data is processed. Formally, denote by n the sample size and by $P_{(r,t)}$ the probability that the r^{th} data is present in the sample after the t^{th} data is processed, where $t > r$. Without loss of generality, we assume that $t \gg n$. We divide the data stream into three intervals: (i) $[1, n]$; (ii) $[n+1, t-n]$; (iii) $[t-n+1, t]$. Note that the probability distributions in these three intervals are different.

When $1 \leq r \leq n$, the case is slightly different from the discussion above. Initially the first n data are sequentially inserted into the sample. So for the first data, $n-1$ of its successors have been inserted into the sample. It is discarded once there is one more data to be selected into the sample from the $(n+1)^{th}$ data to the t^{th} data. For the second data, two more insertions after the first n insertions cause it to be discarded, etc. Thus we can derive the probability formula for the first n data, which is:

$$P_{(r,t)} = \sum_{k=0}^{r-1} Binomial(k; t-n, p),$$

where the function $Binomial()$ represents the Binomial distribution [96]. A binomial function $Binomial(k; m, p_0) = \binom{m}{k} p_0^k (1-p_0)^{m-k}$ calculates the probability that an event happens for k times in m tests, with a probability of p_0 each time. $t-n$ is the total number of remaining data in the stream, k is the number of data selected for the sample and p is the probability.

When $n+1 \leq r \leq t-n$, the situation is similar to the case that we discussed above except that each data initially enters the sample with probability p. So the formula is:

$$P_{(r,t)} = p \times \sum_{k=0}^{n-1} Binomial(k; t-r, p).$$

The last n data are not replaced out once they are selected into the sample, as each of them has less than n successors. Thus the formula is trivial:

$$P_{(r,t)} = p.$$

The complete probability formulae are as follows:

$$P_{(r,t)} = \begin{cases} \sum_{k=0}^{r-1} B(k; t-n, p) & \text{for} \quad 1 \leq r \leq n \\ p \sum_{k=0}^{n-1} B(k; t-r, p) & \text{for} \quad n+1 \leq r \leq t-n \\ p & \text{for} \quad t-n+1 \leq r \leq t, \end{cases} \tag{3.1}$$

where $B()$ is the abbreviation of $Binomial()$.

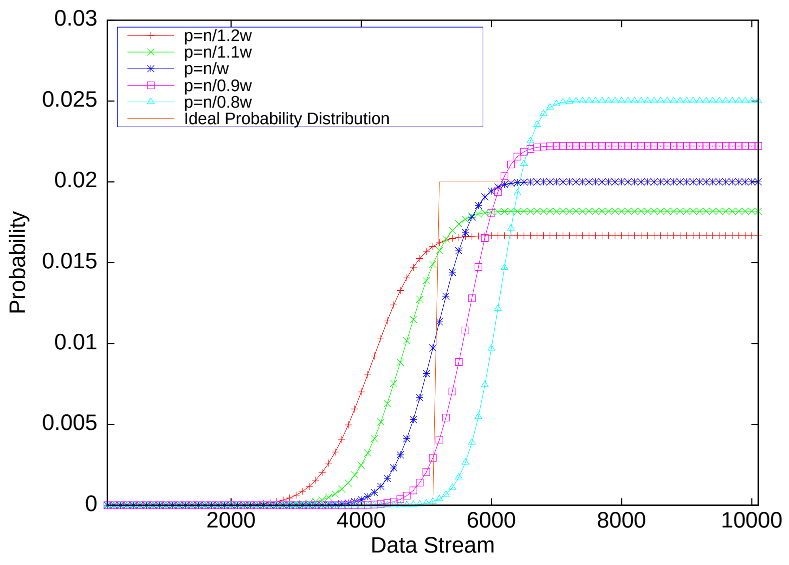


Figure 3.2: Probability of each data to be sampled at $t = 10,000$ by varying p. $n = 100$, $w = 5,000$.

Figure 3.2 shows the probability distributions obtained using the above formulae. In the figure, we vary the value of p and plot the probability for each data to be finally included in the sample at a given time t. The X-axis represents the variable r and the Y-axis is the probability $P_{(r,t)}$. The number of processed data in the figure is $10,000$, the window size is $5,000$ and the sample size is 100. We also plot the corresponding ideal distribution.

In the figure, the line-point graphs show the probability distributions for different values of p according to our analysis, and the dashed graph shows the ideal probability distribution as discussed above. From the graphs, we can see that the probability distribution of $p = n/w$ seems best approximate the ideal distribution. Expired data have lower probabilities to be included in the sample as their age increases while most data in the window have near equi-probability n/w to be selected. The figure suggests that the optimum of this situation is obtained near $p = n/w$. Although equi-probability is only a necessary condition for a sampling algorithm to generate random samples, no further dependency being imposed, it is

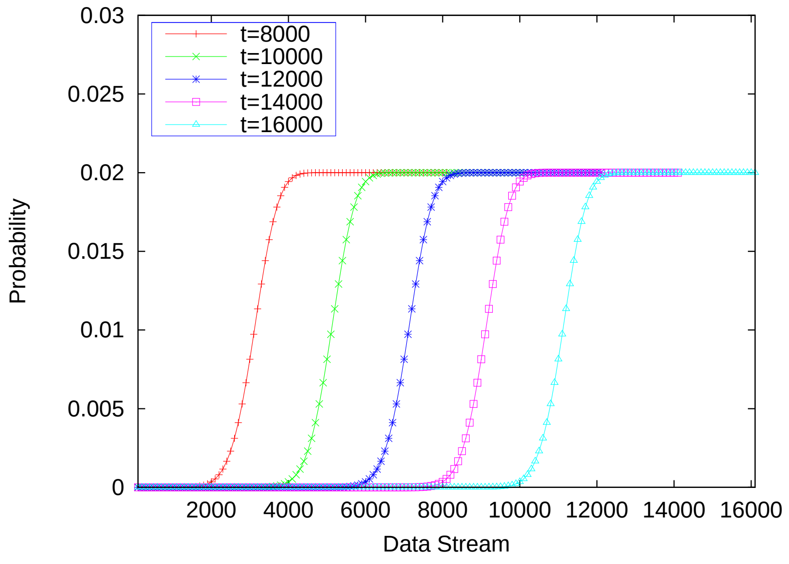


Figure 3.3: Probability of each data to be sampled at $p = n/w$ by varying t. $n = 100$, $w = 5,000$.

clear that it is a sufficient condition for FIFO.

In Figure 3.3, we show, for given n, w and p ($p = n/w$), the probability distributions for selected varying values of t. The graphs being parallel indicates that the same effectiveness is maintained for the successive windows.

3.4 Optimal Inclusion Probability

One important question is the selection of the optimal value of the probability p. That is the value of p that yields the best approximation of a random sampling of the sliding window. We estimate this value in two different ways: analytically and empirically. In Figure 3.2 we see that the probability distributions for different values of p. Among the probabilities, $p = n/w$ seems to be the optimal value. Let us confirm this observation.

We can estimate the optimal value of the probability p by comparing the difference D between the distribution of our algorithm for various values of p and the

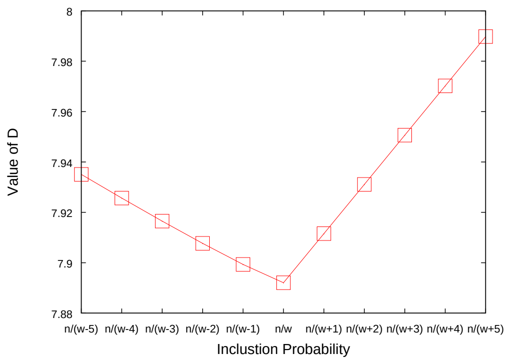


Figure 3.4: Sample 100 data with window size of $5,000$ from a $10,000$ data stream.

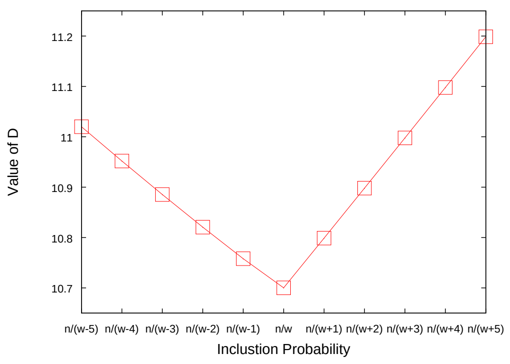


Figure 3.5: Sample 200 data with window size of $2,000$ from a $20,000$ data stream.

ideal distribution. We compute the value of D as follows:

$$D = \sum_{r=1}^{t} |P_{(r,t)} - P_{(r)}|,$$

where $P_{(r,t)}$ can be calculated by Equation3.1 and $P_{(r)}$ is the probability in theory defined as follows:

$$P_{(r)} = \begin{cases} 0 & \text{if} \quad r \leq t - w \\ n/w & \text{otherwise.} \end{cases}$$

Empirically, the optimal value of the probability p should coincide with the smallest value D.

By Experiments, we find that for a given pair of w and n, D always reaches the smaller value when the inclusion probability p approaches to n/w. Thus we believe that the optimal probability should be $\frac{n}{w} \pm d$, where d is a very small real number for adjustment. Figure 3.4 and Figure 3.5 show the results on two sets of parameters.

On the other hand, we analyze the optimal value of p based on the following observes from Figure 3.2 and 3.3. The probability for a data to be included in the sample always firstly increases fast as t gets larger. After a particular point, the increasing ratio becomes smaller and smaller and eventually the probability stays at p. We call the point an inflexion point. For a fixed pair of n and w, we believe that the best approximation of a random sample is obtained when the inflexion point coincides with $t - w + 1$. Below we calculate the value of p that meets the above condition.

Because the distribution is discrete, we cannot calculate the inflexion point by a classical derivation. However, we can use the following approach. Let $d_{(r,t)}$ be the difference between probabilities of the r^{th} data and the $(r+1)^{th}$ data in the sample after the processing of the t^{th} data, that is, $d_{(r,t)} = P_{(r+1,t)} - P_{(r,t)}$. We also define $\Delta d_{(r,t)} = d_{(r,t)} - d_{(r-1,t)}$. If r is an inflexion point, we have that $\Delta d_{(r-1,t)} \geq 0$ and $\Delta d_{(r,t)} \leq 0$. By replacing with the formulae in Section 3.3 (we assume that $n + 1 \leq r \leq t - n$ for the general case), we get,

$$d_{(r,t)} = p \sum_{k=0}^{n-1} [B(k; t-r-1, p) - B(k; t-r, p)]. \tag{3.2}$$

A result in the case of a *Binomial Distribution* [96] states that if $X \sim B(x, p)$ and $Y \sim B(y, p)$ then $X + Y \sim B(x + y, p)$. Thus,

$$\begin{aligned} B(k; t-r, p) &= B(k; t-r-1, p)B(0; 1, p) + B(k-1; t-r-1, p)B(1; 1, p) \\ &= (1-p)B(k; t-r-1, p) + pB(k-1; t-r-1, p). \end{aligned} \tag{3.3}$$

By taking Equation 3.3 into Equation 3.2, we have,

$$\begin{aligned} d_{(r,t)} &= p^2 \sum_{k=1}^{n-1} [B(k; t-r-1, p) - B(k-1; t-r-1, p)] \\ &\quad + p[B(0; t-r-1, p) - B(0; t-r, p)] \\ &= p^2 B(n-1; t-r-1, p). \end{aligned}$$

Thus,

$$\Delta d_{(r,t)} = p^2 [B(n-1; t-r-1, p) - B(n-1; t-r, p)].$$

We can use Poisson distribution, $Poisson(k_0, \lambda) = \lambda^{k_0} e^{-\lambda}/k_0!$ to approximate the Binomial distribution $Binomial(k_0; m, p_0)$, if m is sufficiently large and p_0 is sufficiently small, where $\lambda = mp_0$ [96]. Our formulae satisfy the constraint, so we get,

$$\begin{aligned} \Delta d_{(r,t)} &= p^2 \{ \frac{[(t-r-1)p]^{n-1} e^{-(t-r-1)p}}{(n-1)!} - \frac{[(t-r)p]^{n-1} e^{-(t-r)p}}{(n-1)!} \} \\ &= \frac{p^{(n+1)} e^{-(t-r-1)p}}{(n-1)!} [(t-r-1)^{n-1} - \frac{(t-r)^{n-1}}{e^p}]. \end{aligned}$$

Let $\Delta d'_{(r,t)} = (t-r-1)^{n-1} - \frac{(t-r)^{n-1}}{e^p}$. Obviously, $\Delta d_{(r,t)}$ and $\Delta d'_{(r,t)}$ are of the same positive and negative shape. So if r is an inflexion point, we have $\Delta d'_{(r-1,t)} \geq 0$ and $\Delta d'_{(r,t)} \leq 0$. Because we empirically claim that the inflexion point coinciding with $t - w + 1$ is the optimum, we replace r by $t - w + 1$, so we get,

$$\begin{cases} (w-1)^{n-1} - \frac{w^{n-1}}{e^p} & \geq & 0 \\ (w-2)^{n-1} - \frac{(w-1)^{n-1}}{e^p} & \leq & 0. \end{cases}$$

Thus we get a bound of p, which is,

$$(n-1)\ln\frac{w}{w-1} \leq p \leq (n-1)\ln\frac{w-1}{w-2}. \tag{3.4}$$

An approximation of the bounds can be derived as follows,

$$lower\_bound = (n-1)\ln\frac{w}{w-1} = \ln(\frac{w}{w-1})^{n-1} = \ln(1+\frac{1}{w-1})^{(w-1)\times\frac{n-1}{w-1}}.$$

For a sufficient large w, we can derive the approximation,

$$lower\_bound = \ln(1+\frac{1}{w-1})^{(w-1)\times\frac{n-1}{w-1}} \approx \ln e^{\frac{n-1}{w-1}} = \frac{n-1}{w-1}.$$

Similarly, we can drive the approximation of the upper bound of Equation 3.4, and get the following approximated bounds,

$$\frac{n-1}{w-1} \leq p \leq \frac{n-1}{w-2}. \tag{3.5}$$

For large values of n and w, this bound for the optimal probability is close to n/w, which coincides with the results of the first approach. Thus, we believe that the probability n/w is a good approximation to the optimal probability. Below we use n/w as the inclusion probability for performance evaluation. Table 3.1 illustrates the previous result with some example values of n and w.

n	w	Lower Bound	Upper Bound	$\frac{n}{w}$
500	2000	0.249562	0.249687	0.25
100	2000	0.0495124	0.0495372	0.05
1000	5000	0.19982	0.19986	0.2
1000	12000	0.0832535	0.0832604	0.0833333

Table 3.1: Some bounds calculated by specified n and w.

3.5 Optimizing FIFO

In [112], algorithm Z optimizes algorithm R by periodically skipping some data and directly selecting the next data into the sample. We introduce this idea into our FIFO algorithm and obtain a Z version of FIFO, or for simple, FIFOZ algorithm. Denote by α the number of skipped data. The probability mass function $f(s) = Prob(\alpha = s)$, for $s \geq 0$, is:

$$f(s) = (1-p)^s p,$$

where p is the inclusion probability. So the corresponding cumulative distribution function $F(s) = Prob(\alpha \leq s)$ is:

$$F(s) = \sum_{k=0}^{s} f(k) = 1 - (1-p)^{s+1}.$$

Thus we can generate the independent variable α by first generating an independent uniform variable μ in the interval $[0, 1)$ and then setting α to the smallest s such that $F(s) \geq \mu$, that is:

$$1 - (1-p)^{s+1} \geq \mu.$$

Thus,

$$(1-p)^{s+1} \leq \nu,$$

where ν is also an independent uniform variable in $[0, 1)$ which equals $1 - \mu$. So we get,

$$\alpha = \lceil \log_{1-p} \nu - 1 \rceil.$$

The complete algorithm is given in Algorithm 2.

As a result of periodically skipping data in the stream, FIFOZ algorithm caters

Algorithm 2: FIFOZ Sampling Algorithm

```
Input: $n$ : sample size, $p$ : inclusion probability, $DS$ : data stream
Output: The sample $S$
1 $S \leftarrow \{\}$;
2 Insert sequentially the first n data of DS into S;
3 while NOT EndOFStream(DS) do
4     Randomly generate a number $\nu$ in the interval $[0, 1)$;
5     $\alpha = \lceil \log_{1-p} \nu - 1 \rceil$;
6     Skip the next $\alpha$ data of DS;
7     Insert the next data into S;
8     Discard the oldest data in S;
9 end
```

for sampling data streams of high arrival rate. The expected value of α is:

$$expected(\alpha) = \sum_{k=0}^{+\infty} k(1-p)^k p = \frac{1-p}{p} \tag{3.6}$$

3.6 Performance Evaluation

In this section, we compare FIFO's performance with that of the various sampling algorithms in literatures. We compare the analytical probability distribution and empirically compare the distribution divergence.

The results confirm the inappropriateness of the Reservoir Sampling and Biased Reservoir Sampling algorithms for the problem of sliding windows. They also show that, in practice FIFO performs as effectively as the Simple sliding window sampling (and Random Pairing which degenerates into the Simple algorithm in our case).

3.6.1 Comparison of Analytical Bias Functions

In this section, we analytically compare the probability distribution of FIFO with the optimal probability $p = n/w$ with the probability distributions of the Simple algorithm and the Biased Reservoir Sampling algorithm, respectively. We show that FIFO approximates a random sample without a predictable sample design.

In the Simple algorithm, expired data are discarded and all the data in the sample come from the current window. As the first w data are processed using the Reservoir Sampling algorithm, each of the data have probability n/w to be sampled. Thus, data in the current window also have the optimal sampling probability. However, the simple algorithm is periodical.

In the Biased Reservoir Sampling algorithm [8], the probability of the r^{th} data in the stream being included in the sample after the processing of the t^{th} data is $P_{(r,t)} = e^{-(t-r)/n}$, where n is the sample size.

Figure 3.6 and Figure 3.7 show the results based on two sets of parameters. In Figure 3.6, the case is generating a sample of size 200 from a set of $5,000$ data with window size $1,000$. Figure 3.7 shows the result of sampling from a dataset of $10,000$ data with window size being set to $5,000$ and sample size being set to 500. In both figures, we also plot the probability distribution of Reservoir Sampling. The results demonstrate that the sampling probability of the Biased Reservoir Sampling algorithm suddenly increases to 1 when the stream is close to the end. Data with a little distant history in the window has a very small probability to be included. The Reservoir Sampling algorithm demonstrates a uniform probability distribution over all the data, which is not interesting in our problem. The Simple algorithm has as exactly the same probability distribution as the ideal distribution, despite the sample design is periodical. Obviously, FIFO approximates the ideal distribution better than the Reservoir Sampling algorithm and the Biased Reservoir Sampling algorithm do.

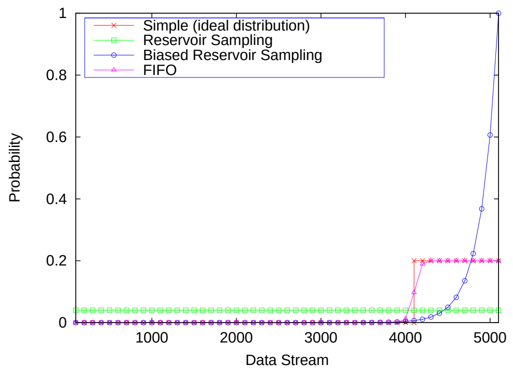


Figure 3.6: Probability distributions of different sampling algorithms. $n = 200$, $w = 1,000$, $p = n/w$.

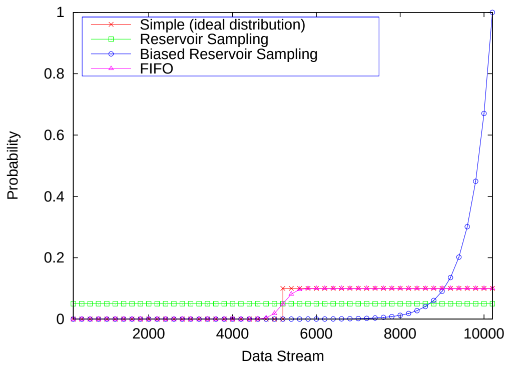


Figure 3.7: Probability distributions of different sampling algorithms. $n = 500$, $w = 5,000$, $p = n/w$.

3.6.2 Empirical Performance Evaluation: Setup

We implement Algorithm R (Reservoir Sampling), Algorithm Z (Skipping Variant of Reservoir Sampling), Random Pairing, the Simple algorithm, the Chain-Sample algorithm, the Biased Reservoir Sampling algorithm and the FIFO algorithm using Java 1.6. For all the implemented algorithms, we consider a sliding window on the stream. All the experiments are run on a dual-core machine with 2.8 GHz CPU and 2 GB memory.

The empirical performance evaluation uses the *Jensen-Shannon Divergence* to quantify the difference between the distributions of the data in the sliding window and in the sample. A small Jensen-Shannon divergence indicates similar distributions. For two distributions $P = \{p_1, p_2 \dots p_n\}$ and $Q = \{q_1, q_2 \dots q_n\}$, Jensen-Shannon divergence measures their similarity as follows,

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M),$$

where $M = \frac{1}{2}(P + Q)$, D_{KL} is the *Kullback-Leibler Divergence*, which is defined as follows,

$$D_{KL}(P||Q) = \sum_{i=1}^{n} p_i \log(p_i/q_i).$$

We use Jensen-Shannon divergence to measure the similarity of distributions of the successive samples with the distribution in the successive sliding windows.

We empirically evaluate the performance of the algorithms with both synthetic and real datasets.

3.6.3 Empirical Performance Evaluation: Synthetic Dataset

The synthetic dataset that we used is a set of $1,000,000$ integers with values ranging from 1 to 10 chosen from a **Zipfian** distribution. In order to create changes in the distribution, we shuffle the distribution every $100,000$ data. The sample size is fixed to $1,000$ and the window size is $50,000$. We plot the graph of the Jensen-Shannon divergence for each algorithm.

The results are shown in Figure 3.8. As we discussed above, Algorithm R and Z produce successive samples of the entire dataset, but not of the sliding windows. As expected, the Jensen-Shannon divergence increases. Indeed the sample in R and Z is representative of the entire stream so far and therefore diverges from the distribution in the window that contains only the most recent data. The reshuffling corresponding to the changes in distribution are clearly visible on the plot. The Chain-Sample algorithm produces low Jensen-Shannon divergence values. The Biased Reservoir Sampling algorithm generates very high peaks at the interfaces of two intervals with different distributions. It is too sensitive to the changes, as expected as well. The Simple algorithm performs the best in the above algorithms. The Random Pairing algorithm degenerates into the Simple algorithm. Our FIFO algorithm performs similarly to the Simple algorithm, which can be seen clearly in Figure 3.9.

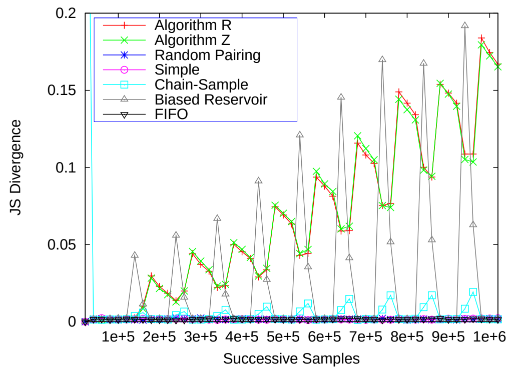


Figure 3.8: The Jensen-Shannon divergence values of successive samples and sliding windows for all the algorithms.

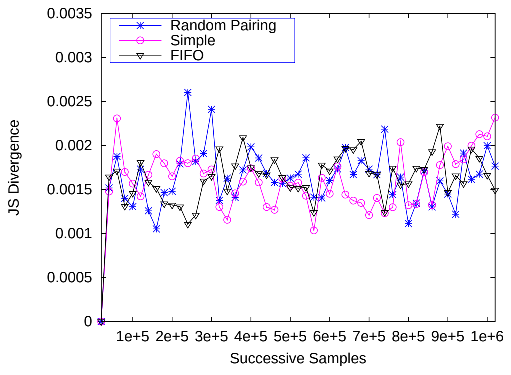


Figure 3.9: Comparison of FIFO, RP and simple algorithm, 10 datasets of value range from 1 to 10.

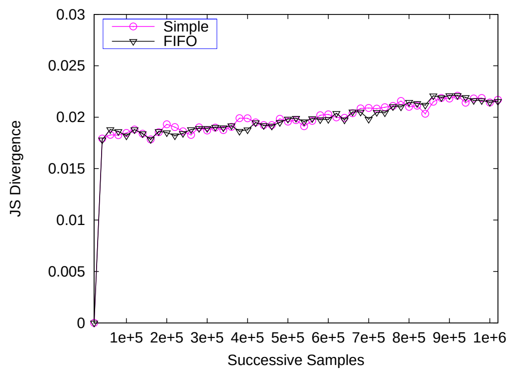


Figure 3.10: Comparison of FIFO and simple algorithm, 100 datasets of value range from 1 to 100.

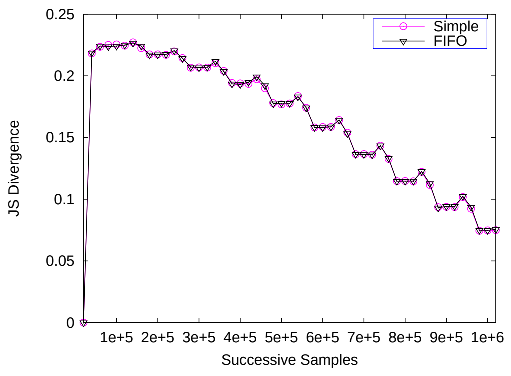


Figure 3.11: Comparison of FIFO and simple algorithm, 100 datasets of value range from 1 to 1,000.

We then extend our experiment to further compare the performances of FIFO with that of the simple algorithm. Figure 3.10 and Figure 3.11 show the results of the experiments on 100 datasets containing integers with values ranging from 1 to 100 and 100 datasets containing integers with values ranging from 1 to 1,000 respectively. The reader remembers that, although the two algorithms perform equally well, the simple algorithm sample design is periodical.

We also confirm the optimal value for p by evaluating FIFO's performance by setting different inclusion probabilities. Figure 3.12 and Figure 3.13 show the

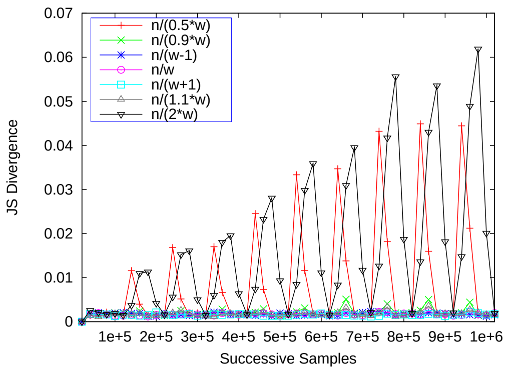


Figure 3.12: Comparison of FIFO with different inclusion probabilities, 10 datasets of value range from 1 to 10, $w = 50,000$, $n = 1,000$.

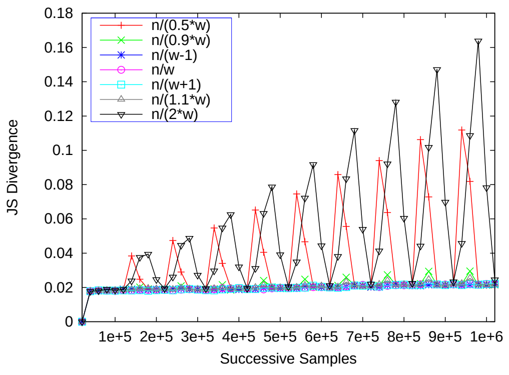


Figure 3.13: Comparison of FIFO with different inclusion probabilities, 100 datasets of value range from 1 to 100, $w = 50,000$, $n = 1,000$.

results on the 10 datasets of value from 1 to 10 and 100 datasets of value from 1 to 100 respectively. From the figure, we can see that the optimal probability should near n/w, which coincides with the results in Figure 3.2.

3.6.4 Empirical Performance Evaluation: Real Dataset

The real life dataset that we used in the experiments is the weather data collected at [46] which records the surface synoptic weather information for the entire globe from December 1981 to November 1991. The reports come from land stations and ships located in particular positions of the earth. We use the reports from land stations in 1990, and we are interested in the current weather attribute which has a domain of 48 integer values. The data is sorted in chronological order. The total number of the records is $1,344,024$. We set the sample size to be $1,000$ and the window size to be $50,000$. We still calculate successive Jensen-Shannon divergence values as in the synthetic dataset experiment to evaluate the performances of the algorithms. The results are shown as Figure 3.14. We can see that Algorithm R, Algorithm Z and the Biased Reservoir Sampling algorithm produce higher Jensen-

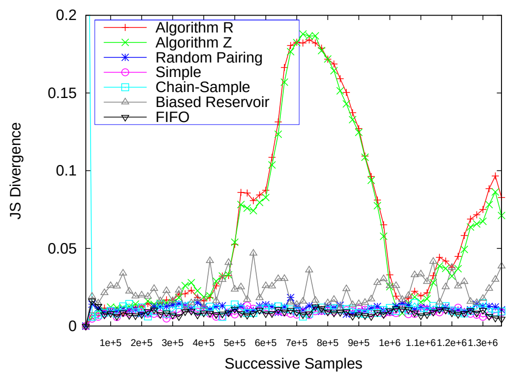


Figure 3.14: Performance evaluation on real life dataset.

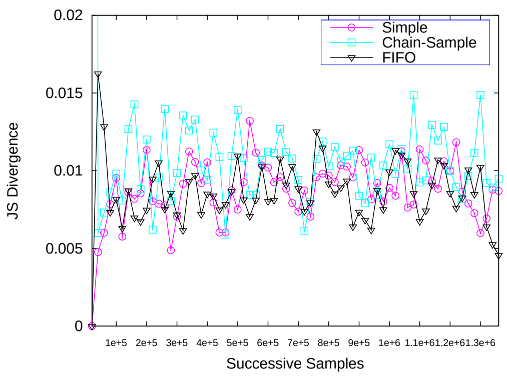


Figure 3.15: Performance evaluation on real life dataset.

Shannon divergence values. The other four algorithms perform relatively better. However, Random Pairing degenerates into the Simple algorithm. In Figure 3.15, we can see their performances more clearly. The Chain-Sample algorithm produces a slightly higher Jensen-Shannon divergence than FIFO and the Simple algorithm. FIFO and the Simple algorithm are still the best.

3.6.5 Empirical Performance Evaluation: Efficiency

We also run these algorithms on data streams of different lengths to evaluate and compare their running time. The stream lengths are set to $1,000,000$, $5,000,000$, $10,000,000$, $50,000,000$, $100,000,000$ and $500,000,000$. Figure 3.16 and Figure 3.17 show the results on two sets of sampling parameters. The window sizes are $50,000$ and $100,000$ and the sample sizes are $1,000$ and $5,000$, respectively. We see that the running times of Reservoir Algorithm R, Random Pairing, Biased Reservoir Sampling, FIFO are higher than that of the other algorithms. This is because these four algorithms have to process every data encountered in the stream. The optimized variant of FIFO, the FIFOZ Sampling algorithm, is as fast as the Chain-Sampling algorithm. However, the exact way of sampling n data with Chain-Sampling is still unknown. In section 3.5 we give the expected value of α (the value

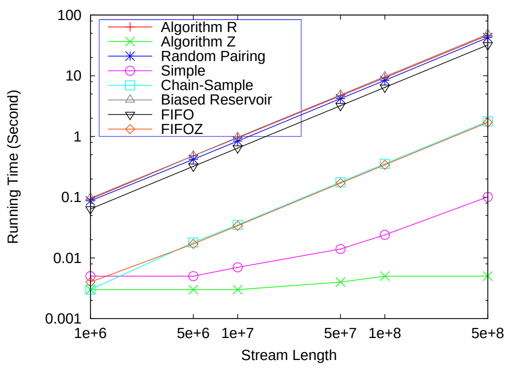


Figure 3.16: Running time evaluation, $w = 50,000$ and $n = 1,000$.

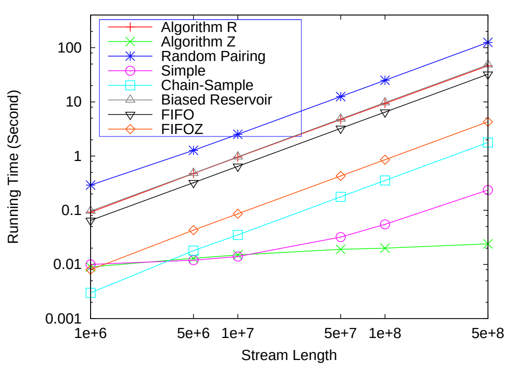


Figure 3.17: Running time evaluation, $w = 100,000$ and $n = 5,000$.

of skipping of FIFOZ) in Equation 3.5. For $p = \frac{n}{w}$, we get $expected(\alpha) = \frac{w-n}{n}$ which is a constant for given n and w. Therefore the running time of FIFOZ increases linearly with the increasing of the length of the data stream. On the other hand, we know that the expected value of α of Algorithm Z is $\frac{t-n+1}{n-1}$ [112], where t is the processed data. The expected value suggests that the value of skipping of Algorithm Z increases linearly along with the length of the stream. In other word, the running time of Algorithm Z should remain stable, despite the increasing of the data stream length. The results in the figure confirm the intuition. We observe that the running time of Algorithm Z remains almost unchanged in spite of the increasing of the stream length, whereas the running time of FIFOZ increases linearly with the stream length. We also implement the skipping variant of the Simple algorithm. In the skipping variant, the window is moved each time directly to the next data that is selected into the sample to replace the expired data. The number of skipped data is equal to the distance between the oldest data and the second oldest data in the current sample. The expected value of this distance is $\frac{w-n}{n}$ as each data is selected uniformly at random from the current window. The design of the Simple algorithm makes it periodically skip some data, whereas in FIFOZ, we have to compute the values of skipping with random number generation and the logarithm computation. Therefore FIFOZ runs slightly slower than the Simple algorithm. Nevertheless, FIFOZ improves significantly the efficiency of FIFO, especially when $w \gg n$.

3.7 Summary

In this work, we propose a new sampling algorithm, called FIFO sliding window sampling or FIFO, for short, for sampling sliding windows over data stream.

We compare its performance to that of existing stream and sliding window sampling algorithms. We analyze the properties of FIFO analytically and empirically to show that our new algorithm is effective: it can maintain a near random sample of a sliding window with fixed memory and without reproducing its sample design, with the optimal selection of the inclusion probability of each data.

FIFO is also very efficient as it only maintains a queue and samples each data once with a bernoulli process. We also adopt the skipping idea of Algorithm Z and implement the FIFOZ sampling algorithm. The results show FIFOZ improves the efficiency significantly.

Although we have shown empirically and argued analytically that FIFO is most effective for p near n/w, we are now trying to obtain an exact analytical formula for this optimum.

Chapter 4

Sampling Connected Induced Subgraphs Uniformly at Random

4.1 Introduction

The versatility of graphs makes them an almost universal data structure in domains as varied as social network, transportation and bioinformatics, to cite a few among the obvious. The challenge for modern applications is the effective and efficient processing of very large graphs. One way to circumvent scalability issues arising from this challenge is to replace the processing of very large graphs by the processing of representative subgraphs of manageable size. In this work, we study a new graph sampling problem.

We observe that the graphs of interest are often required to be connected (or that one is interested in connected components). This is the case, for instance, when one is looking for graphlets and motifs in applications such as graph pattern mining (see [61, 94], for example). This is also the case when one is studying social networks where communities are characterized by their connectivity. In this domain connected induced subgraphs are the preferred basis of studies of network topology (see [10, 66], for example) and evolution (see [110], for example), for instance.

For statistical analysis, data mining and simulation applications a common requirement for a subgraph construction or sampling algorithm is that it maintains graph properties. A uniformly sampled connected induced graph has the advantage that it naturally statistically maintains local properties such as vertex degrees and clustering coefficients among others. In addition a strong guarantee on the distribution, such as uniformity, from which the graph is sampled is very useful for statistical analysis, simulation applications and randomized algorithms.

Motivated by these observations, we study the uniform (or simple) random sampling of a connected induced subgraph of a prescribed size from a graph.

For the sake of simplicity, and without loss of generality as far as the algorithms discussed are concerned, we consider *simple graphs* of the form $G =< V, E >$, where V is a set of vertices and E is a set of pairs of vertices called edges. The *size* (also called *order*) of a graph $G =< V, E >$ is the number of its vertices, $\mid V \mid$. A *subgraph* of a graph $G =< V, E >$ is a graph $G' =< V', E' >$ such that $V' \subseteq V$ and $E' \subseteq E$. A subgraph of a graph G is *induced* if it contains all the edges that appear in G between any two of its vertices.

Definition 1. *Let $G =< V, E >$ be a graph. A subgraph $G' =< V', E' >$ of G is* induced *if and only if:*

$$\forall x \in V' \ \forall y \in V' \quad \{x, y\} \in E \Rightarrow \{x, y\} \in E'$$

The problem we am studying in this work is the **generation of connected induced subgraphs of size k uniformly at random from a connected graph G of size n, where $1 \leq k \leq n$.**

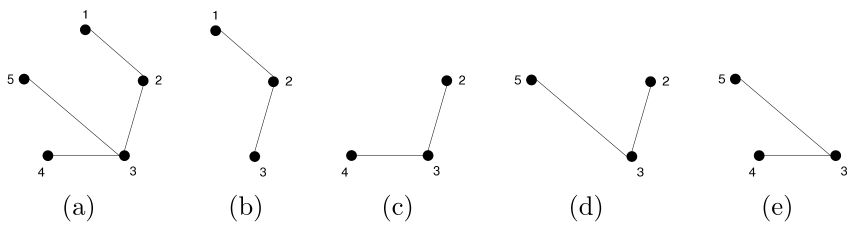


Figure 4.1: A connected graph and its connected induced subgraphs

Example 1. *The graph in Figure 4.1(a) is a simple graph of size* 5*. It has* 4 *edges. There are* 4 *connected induced subgraphs with* 3 *vertices shown in Figure 4.1(b), 4.1(c), 4.1(d) and 4.1(e), respectively. We ought to devise algorithms able to randomly generate each of this connected induced subgraphs with equi-probability, i.e.* $\frac{1}{4}$ *in this example.*

We devise, present and discuss four algorithms that leverage different techniques: Rejection Sampling, Random Walk and Markov Chain Monte Carlo.

The first algorithm, which we call Acceptance-Rejection Sampling (ARS) is a trivial variation of the standard Rejection Sampling [69]. It serves as a baseline reference as it is guaranteed to be a uniform random sampling.

The second algorithm, which we call Random Vertex Expansion (RVE), adapts the idea of a random walk on the original graph to the gradual construction of the sample.

The third algorithm, which we call Metropolis-Hastings Sampling (MHS), is a Markov Chain Monte Carlo algorithm. The general idea of algorithms from this known and generic family is a random walk on a graph of sample states. The effectiveness of such algorithms is guaranteed provided a sufficient duration, *mixing time*, of the random walk.

The fourth algorithm, which we call Neighbour Reservoir Sampling (NRS) is our main contribution. It operates on the same Markov Chain as MHS but tries to avoid local computation of the degrees of states and long or unbound random walks, and adjust the bias of RVE by combining the idea of expansion from RVE with the idea of reservoir from Reservoir Sampling [112].

Notice again that, differently from these existing algorithms that aim to find a subgraph most similar to the original graph for some property, we consider the random generation of connected induced subgraphs of prescribed size uniformly at random from a connected simple graph G.

4.2 The Algorithms

We first introduce a baseline algorithm based on Rejection Sampling [69]. We then revisit the algorithm of [61] for sampling connected subgraphs. Finally, we propose two non-trivial algorithms for practically sampling the subgraphs. The two algorithms leverage the idea of traversing a Markov chain of connected induced subgraphs.

4.2.1 Acceptance-Rejection Sampling

The simplest algorithm to sample, uniformly at random, an induced subgraph of size k from an original graph of size n is to select k vertices uniformly at random, complete the induced graph by adding all the edges linking the vertices and then check for connectivity of the induced subgraph. If the induced subgraph is not connected then it is rejected and the selection restarts. If the induced subgraph is connected then it is accepted.

It is a simple instance of Rejection Sampling (see [69], for instance). We call this algorithm *Acceptance-Rejection Sampling* (ARS). The pseudo-code for ARS is given in Algorithm 3.

Notice that the same approach can be applied to generate uniformly at random an induced subgraph with any desired property other than connectivity.

Algorithm 3: Acceptance-Rejection Sampling

```
Input: G : the original graph with n vertices, k : subgraph size
Output: a connected induced subgraph g with k vertices
1 do
2     Select k vertices uniformly at random from G;
3     Generate the induced graph g of the k vertices;
4 while g is not connected;
5 Return g
```

Proposition 1. *Given a graph G of size n and an integer k where $n \geq k$, the ARS algorithm generates a connected induced subgraph of size k (if it exists) from G uniformly at random.*

Proof. Let S be the set of induced subgraphs of size k in G. Let C be the set of connected induced subgraphs of size k in G. Let g be the probability distribution of S. Let f be the probability distribution of C. In a uniform distribution, the probability density function (pdf) of a subgraph $c \in C$ is $f_{(c)} = 1/|C|$ and $g_{(c)} = 1/|S|$. Let $M = |S|/|C|$. For each c generated from the distribution g, we accepted it with probability $f_{(c)}/(M \times g_{(c)}) = 1$ if c is connected; otherwise, we reject c. The accepted subgraph c follows the distribution f, i.e., it is generated uniformly at random from C. □

ARS is a baseline algorithm that provides a reference for effectiveness. It is a rather brute-force algorithm. We expect ARS to be efficient for dense graphs but otherwise generally inefficient when connected induced subgraphs are a small fraction of induced subgraphs.

4.2.2 Random Vertex Expansion

One natural method to generate connected subgraphs is to explore a graph from a starting vertex moving gradually and randomly to neighbouring vertices. This generalization of a random walk, which we call *Random Vertex Expansion* (RVE) has been independently proposed and used by many authors. In particular, Kashtan et al. in [61] use it to sample network motifs. Variants of RVE can be implemented with different random selection of the next vertex. We have experimented with several such selection functions and, for the sake of simplicity, only report here the most relevant one. Namely, in the variant we are discussing, RVE chooses the next vertex by selecting uniformly at random one of edges connecting a vertex of the current subgraph with a vertex not yet in the subgraph. The algorithm terminates when the subgraph has the desired size. The pseudo-code for RVE is given in Algorithm 4.

Algorithm 4: Random Vertex Expansion

```
   Input: $G$ : the original graph with $n$ vertices, $k$ : subgraph size
   Output: a connected induced subgraph $g$ with $k$ vertices
 1 $E \leftarrow \emptyset$;
 2 Select uniformly at random an edge $e$ of $G$;
 3 $E \leftarrow e$;
 4 while $E$ contains less than $k$ vertices do
 5     $EL \leftarrow \emptyset$;
 6     $EL \leftarrow$ edges connecting a vertex of $E$ with a vertex not yet in $E$;
 7     Select uniformly at random an edge $e$ from $EL$;
 8     $E \leftarrow e$;
 9 end
10 Return the induced graph $g$ of $E$;
```

The probability of sampling a $k-$vertex subgraph is then the sum of the probabilities of the permutations of its $k-1$ edges. It is given by Equation 4.1 where S_m is the set of all the valid permutations of $k-1$ edges to sample a certain subgraph,

E_j is the j^{th} edge in an $(k-1)$-edge permutation.

$$P = \sum_{\sigma \in S_m} \prod_{E_j \in \sigma} Pr[E_j = e_j | (E_1, E_2, \ldots, E_{j-1}) = (e_1, e_2, \ldots, e_{j-1})]. \quad (4.1)$$

The more edges a connected induced subgraph of size k has, the more permutations of its $k-1$ edges, and the higher its probability to be selected. RVE has bias to those subgraphs that are denser, i.e., have higher average clustering coefficient. The main cost of RVE is computing the list of edges connecting a vertex of the current subgraph with a vertex not yet in the subgraph in each step. Suppose the maximal degree of vertices in G is D, the worst complexity is $O(Dk)$. The complexity can be reduced to $O(k^2)$ by maintaining an efficient data structure [61].

4.2.3 Metropolis-Hastings Sampling

Another idea for catering for the requirement of connectivity is to sample on an ergodic *Markov chain* whose states represent all the connected induced subgraphs with prescribed size of a given original graph. The approach belongs to the *Markov Chain Monte Carlo* (MCMC) family [41]. An MCMC sampling algorithm constructs and randomly traverses a Markov chain whose states are all the candidate samples, starting from any initial state of the Markov chain. After sufficiently many steps, this random walk converges and each state is visited with probability proportional to the degree of that state. The number of random walk steps needed for convergence is called the *mixing time*. The corresponding probability distribution is called *stationary distribution* and the probability is called *stationary probabilities*.

For the problem at hand, the states of the Markov chain are the connected induced subgraphs g of size k of the graph G. Two states g_i and g_j are neighbours of each other if and only if $V(g_i) - V(g_j) = \{v_i\}$ and $V(g_j) - V(g_i) = \{v_j\}$, where $V(g_i)$ is the vertex set of g_i, that is, one can get g_j by deleting v_i from g_i and adding v_j to it, and vice versa. This Markov chain is constructed while it is traversed. The starting state of the random walk can be any connected induced subgraph g_0 of size k. In order to construct such a graph it suffices, for instance, to start at a random vertex and to traverse, by any convenient means, the graph G until we visit k vertices.

In each step, the random walk may select the next state uniformly at random from the neighbours of the current state and transfers to it. We can see that all the connected induced subgraphs can be reached from any initial state by a sequence of replacement of the vertices, which indicates the Markov chain is irreducible. Moreover, rejecting the transition means adding self-loops to the states, which makes the Markov chain aperiodic. Therefore our Markov chain is ergodic and has a stationary distribution.

However, the stationary probability of each state is proportional to its degree in the Markov chain. Consequently for the Markov chain that we are considering, the distribution is not necessarily uniform. In Section 6 where we generate random graphic sequences, we produce the uniform distribution by introducing intermediate states into the Markov chain. Another option to adjust this bias is by using the *Metropolis-Hastings* algorithm [52]. Each step from g_i to g_j is accepted with probability $\frac{d_i}{d_j}$ where d_i is the degree of g_i. This approach is actually balancing the probability of visiting the states with their different degrees. One may notice that if $d_i > d_j$ the transition is accepted definitely, whereas the smaller d_i compared to d_j the higher the chance that the random walk stays at g_i.

We call this algorithm *Metropolis-Hastings Sampling* (MHS). The pseudo-code for MHS is given in Algorithm 5.

Algorithm 5: Metropolis-Hastings Sampling

```
Input: $G$ : the original graph with $n$ vertices, $k$ : subgraph size, $t$ : the
       number of random walk steps
Output: a connected induced subgraph $g$ with $k$ vertices
1  Perform any graph traverse algorithm on $G$ to get an initial connected
   induced subgraph $g$ of size $k$;
2  while $t > 0$ do
3      Select $g'$ uniformly at random from the neighbors of $g$;
4      Generate a random number $\alpha \in [0, 1)$;
5      if $\alpha < \frac{d_g}{d_{g'}}$ then
6          $g = g'$
7      end
8      $t = t - 1$
9  end
10 Return $g$;
```

The Markov chain and the degree of the states are computed during the traversal. At each step the algorithm only needs to compute the degree of the selected neighbour of the current state and temporarily store the neighbour information of that neighbour. If the transition is accepted, it moves to the next state and directly uses the information computed in previous step, and then iteratively computes and stores the information of the next selected neighbour, etc. In this way the memory usage is bounded by the largest degree of a state in the Markov chain. In terms of time complexity, suppose on average each connected induced subgraph of size k has l edges and m neighbour vertices, MHS needs on average $O(km(k + l))$ time to compute its neighbours states in the Markov chain, where $O(k + l)$ is the time complexity for checking the connectivity. Suppose on average each connected induced subgraph has d neighbour states, then each local computation of the degree of neighbour states takes $O(dkm(k+l))$ time. Therefore the overall time complexity of MHS is $O(tdkm(k + l))$, where t is the number of random walk steps.

One crucial issue for Markov Chain Monte Carlo algorithms is to determine the mixing time. When the mixing time cannot be determined analytical, one can use statistical tests of convergence [31]. One simple and practical such test is the *Geweke* diagnostics [39]. The basic idea is that for a sequence of random walk iterations, Geweke diagnostics compares the distribution of some metric of interest between the beginning part and the ending part of the sequence. As the random walk iterates, the correlation between the two parts decreases and thus the distribution of the metric of interest should become identical. Geweke diagnostics defines the $z-$score such that $z = \frac{E(m_b)-E(m_e)}{\sqrt{Var(m_b)+Var(m_e)}}$, where m_b and m_e denote the metric of interest in the beginning part and the ending part of the Markov sequence, respectively. Typically the beginning part is defined as the first 10% part and the ending part is defined as the last 50% part. Then multiple chains start from different initial states. The convergence is declared when the z-scores of all the chains fall into the range $[-1, 1]$ with a mean of 0 and a variance of 1. Below we empirically evaluate the convergence of MHS using the Geweke diagnostics.

4.2.4 Neighbour Reservoir Sampling

In order to adjust the bias of RVE, we consider a method that decreases the probability to sample the induced subgraphs with higher average clustering coefficient. We sample the first k vertices using RVE and get a subgraph g_k. After that, we continue to choose the i^{th} vertex v_i by selecting uniformly at random one of edges connecting a vertex of g_{i-1} with an unprocessed vertex. Then we insert v_i into g_{i-1} with probability $\frac{k}{i}$. In case of a success, one vertex of g_{i-1} is replaced by v_i uniformly at random and we get a new subgraph g_i. If g_i is connected, we keep it; otherwise, $g_i = g_{i-1}$. We iteratively select the new vertices until there is no such an edge that connects a vertex of g_i with an unprocessed vertex.

We call this algorithm *Neighbour Reservoir Sampling* (NRS) as the algorithm samples with a reservoir and always chooses the new vertices from the neighbours. The pseudo-code for NRS is given in Algorithm 6.

Algorithm 6: Neighbour Reservoir Sampling

```
Input: $G$ : the original graph with $n$ vertices, $k$ : subgraph size
Output: a connected induced subgraph $g$ with $k$ vertices
$V \leftarrow$ the vertices selected using RVE;
$EL \leftarrow$ edges connecting a vertex of $V$ with an unprocessed vertex;
$i = k$;
while $EL$ is not empty do
    $i++$;
    Select uniformly at random an edge $e$ from $EL$;
    $v =$ the unprocessed vertex of $e$;
    Generate a variable $\alpha \in [0, 1)$ uniformly at random;
    if $\alpha < k/i$ then
        Select uniformly at random a vertex $u$ from $V$;
        $V' \leftarrow V \backslash u \cup v$;
        if the induced subgraph of $V'$ is connected then
            $V \leftarrow V'$
        end
    end
    Recompute $EL$ of the current subgraph
end
Return the induced graph $g$ of $V$;
```

The algorithm begins with standard RVE and continues to select new vertices using the same strategy as RVE. However, the vertices with higher local clustering coefficient have higher probability to be replaced, because by deleting them the subgraphs have higher probability to remain connected. The bias to the subgraphs having higher average clustering coefficient is therefore adjusted by this mechanism. The new vertices enter the subgraph with a decreasing probability so that each selected vertex can be sampled with the same probability [112] despite the connectivity of the graph.

In addition, Similarly to MHS, NRS traverses a Markov chain of connected induced subgraphs of size k. However, NRS takes at most $n-k$ steps, and in each step, NRS checks the connectivity of the possible next state at most once. In each step, NRS maintains a list of edges connecting one unprocessed vertex and one vertex in the current subgraph. This process takes $O(m_e)$ time, where m_e is the number of edges connecting to the current subgraph. Therefore the overall time complexity is $O(Dk + (n-k)(m_e + k + l))$, where $O(Dk)$ is the time to compute the first subgraph of size k, $O(k+l)$ is the time to check the connectivity of each subgraph.

As a result, NRS adjusts the bias of RVE as well as avoids the local computation of the degrees of Markov chain states and the long or unbound random walks of MHS.

4.3 Performance Evaluation

4.3.1 Experimental Setup

We implement the four algorithms in C++ and run them on an Intel Core 2 Quad machine with Ubuntu 10.4 and 4GB main memory.

We conduct experiments with both synthetic datasets and real life datasets. First, we evaluate and compare the performance of the proposed algorithms with synthetic graphs generated in the Erdős-Rényi and Barabási-Albert models. We generate graphs of varying size and density. These properties can be controlled directly or indirectly by the parameters of the two models. Second, we show that induced subgraphs that are uniformly sampled can preserve significant properties of the original graph. We collect four real life graphs from *SNAP* [6] and one real life graph from *arXiv.org*. Table 4.1 lists the basic statistics of the five graphs. We sample a series of subgraphs with incremental sizes from these real graphs using NRS and calculate average errors on multiple graph properties between the subgraphs and the original graphs, over all the five graphs.

Dataset	#Vertices	#Edges	Description
HEP-TH	$29,555$	$352,807$	A citation graph from e-print arXiv
HEP-PH	$30,567$	$348,721$	A citation graph from e-print arXiv
AS	$6,474$	$26,467$	A autonomous system consisting of routers
Bipar-arXiv	$57,381$	$133,170$	A bipartite graph from arXiv describing the affiliation between authors and papers
Trust	$75,879$	$508,960$	A who-trusts-whom graph from `epinions.com`

Table 4.1: Description of the real life datasets.

We preliminarily discuss the convergence of MHS using Geweke diagnostics.

We comparatively evaluate the effectiveness of the four algorithms by measuring the standard deviation from the uniform distribution and by comparing the average subgraph properties.

We comparatively evaluate the efficiency of the four algorithms by measuring their execution time.

We comparatively evaluate the overall performance of the four algorithm in terms of the efficiency versus the effectiveness.

We evaluate the property-preserving of uniform sampling by measuring the Kolmogorov-Smirnov D-statistic on different graph criteria between the subgraphs and the original graphs. The D-statistic is used to compare two distributions, even if they are of different scalings. It is defined as $D = \max_x |F'_{(x)} - F_{(x)}|$, where $F'_{(x)}$ and $F_{(x)}$ are the empirical distribution functions and x is over the range of random variable of the distribution. $F_{(x)}$ for n *iid* observations x_i is defined as $F_{(x)} = \frac{1}{n}\sum_{i=1}^{n} I_{x_i \le x}$, where $I_{x_i \le x}$ is equal to 1 if $x_i \le x$ and equal to 0 otherwise. A lower value of D-statistic indicates higher similarity of two distributions.

4.3.2 Mixing Time

We empirically evaluate the mixing time of MHS by detecting its convergence using Geweke diagnostics.

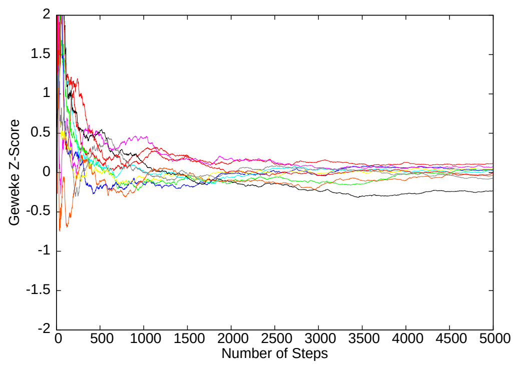


Figure 4.2: Geweke diagnostics for a Barabási-Albert graph with 1000 vertices and $p = 0.1$. The sampled subgraph size is 10. The metric of interest is average degree.

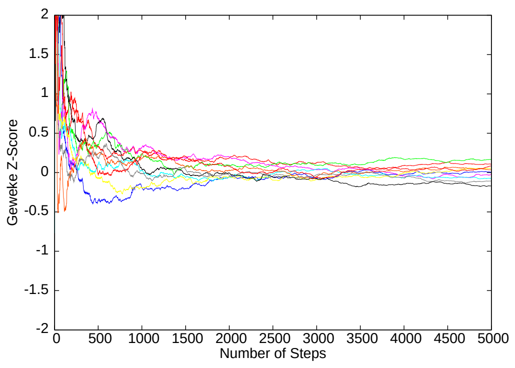


Figure 4.3: Geweke diagnostics for a Barabási-Albert graph with 1000 vertices and $p = 0.1$. The sampled subgraph size is 10. The metric of interest is average clustering coefficient.

We evaluate the convergence of MHS with an Erdős-Rényi graph with $1,000$ vertices with $p = 0.1$, and with a Barabási-Albert graph with 500 vertices and 10 new links per new vertex[1]. We sample connected induced subgraphs with prescribed size 10. For each graph, we run 10 chains from different starting vertex. We compute the z-score of Geweke diagnostics for each chain after every random walk step, using the metric of average degree and average clustering coefficient of the subgraphs.

The results are presented in Figures 4.2, 4.3, 4.4 and 4.5.

We see that for the Erdős-Rényi graph, the z-scores have a mean 0 and a variance less than 0.5 after 2000 random walk steps. For the Barabási-Albert graph, this number of random walk steps is around 1500. We then use this empirically results in the evaluation below.

[1]We denote by d this parameter, i.e., $d = 10$. Below we use this form.

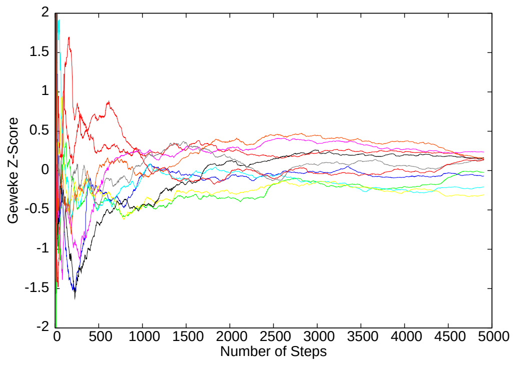


Figure 4.4: Geweke diagnostics for a Barabási-Albert graph with 500 vertices and $d = 10$. The sampled subgraph size is 10. The metric of interest is average degree.

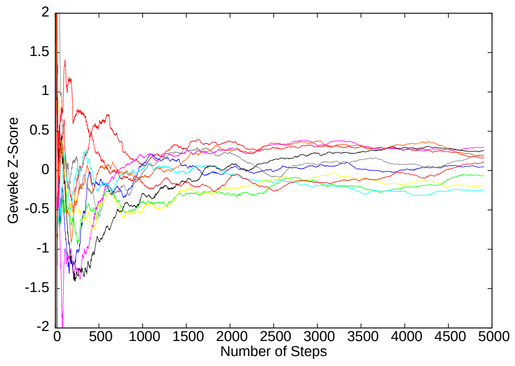


Figure 4.5: Geweke diagnostics for a Barabási-Albert graph with 500 vertices and $d = 10$. The sampled subgraph size is 10. The metric of interest is average clustering coefficient.

4.3.3 Effectiveness

We evaluate the effectiveness of the algorithms on small and large graphs, respectively.

4.3.3.1 Small Graphs

We measure the standard deviation of the four algorithms with four Barabási-Albert graphs with 15 vertices and $d = 1, 2, 3, 4$, respectively. We sample connected induced subgraphs with prescribed sizes varying from 1 to 15. For each size, we generate 10 samples on average for every distinct induced subgraphs.

The results are presented in Figures 4.6, 4.7, 4.8 and 4.9.

We see that ARS and MHS yield the lowest overall standard deviation while NRS remains competitive unless the graph is sparse. RVE does not perform as well as the above three algorithms because it has bias to subgraphs with higher average degree and higher average clustering coefficient.

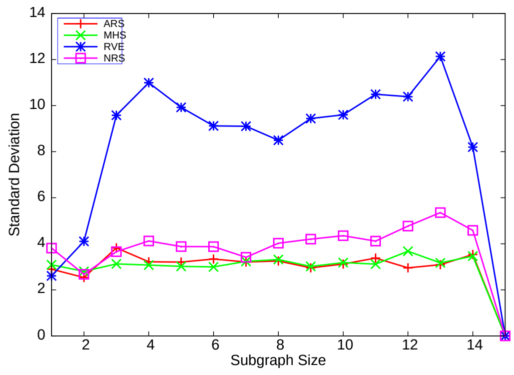


Figure 4.6: Standard deviation from uniform distribution. The Barabási-Albert graph has 15 vertices and $d = 1$.

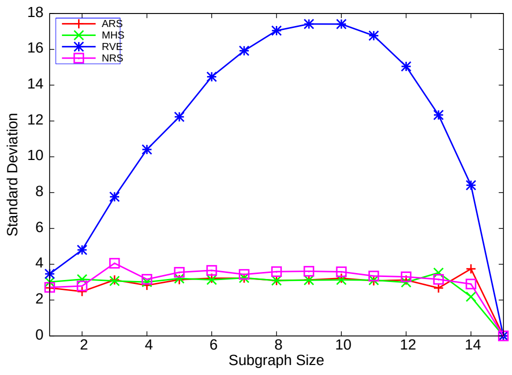


Figure 4.7: Standard deviation from uniform distribution. The Barabási-Albert graph has 15 vertices and $d = 2$.

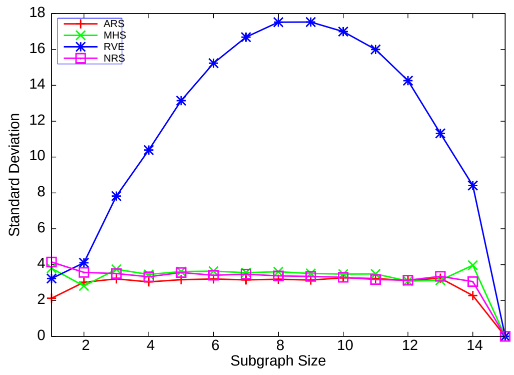


Figure 4.8: Standard deviation from uniform distribution. The Barabási-Albert graph has 15 vertices and $d = 3$.

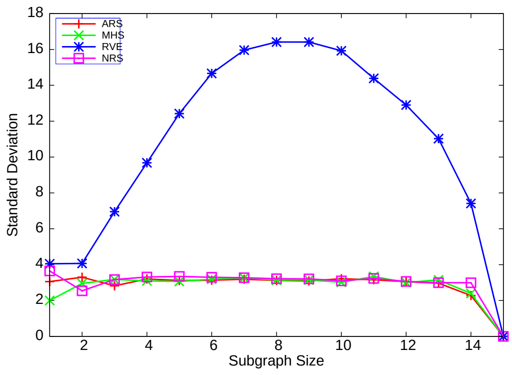


Figure 4.9: Standard deviation from uniform distribution. The Barabási-Albert graph has 15 vertices and $d = 4$.

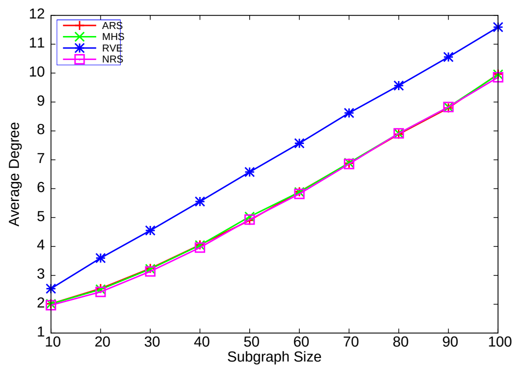


Figure 4.10: Comparison of average degree of samples. The original Erdős-Rényi graph has 1000 vertices and $p = 0.1$.

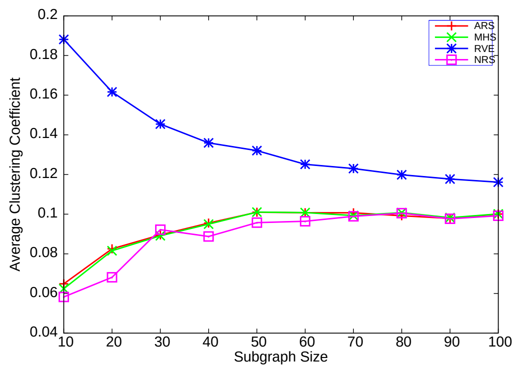


Figure 4.11: Comparison of average clustering coefficient of samples. The original Erdős-Rényi graph has 1000 vertices and $p = 0.1$.

4.3.3.2 Large Graphs

It is impractical to generate all the connected induced subgraphs for large graphs. We therefore turn to compare average graph properties of connected induced subgraphs sampled by different algorithms.

We measure the average degree and average clustering coefficient of the subgraphs generated by the four algorithms with the Erdős-Rényi graph of 1000 vertices with probability 0.1 and with the Barabási-Albert graph of 500 vertices and $d = 10$, which are used in Section 4.3.2. We sample connected induced subgraphs with prescribed sizes varying from 10 to 100 in increments of 10. For each size, we sample 100 connected induced subgraphs and calculate the average of average degree and average clustering coefficient of these subgraphs.

The results are presented in Figures 4.10, 4.11, 4.12 and 4.13.

We see that ARS and MHS always coincide with each other on both properties while NRS remains competitive. RVE is biased towards subgraphs with higher average degree and higher average clustering coefficient. This is because RVE tends to sample denser subgraphs.

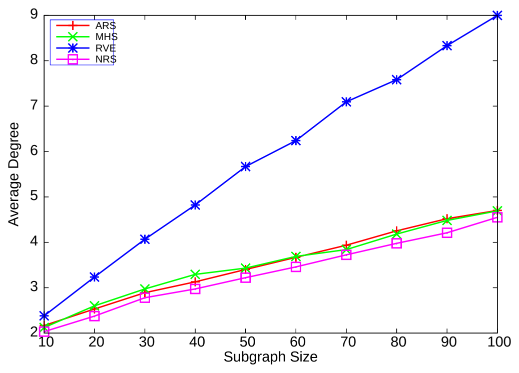


Figure 4.12: Comparison of average degree of samples. The original Barabási-Albert graph has 500 vertices and $d = 10$.

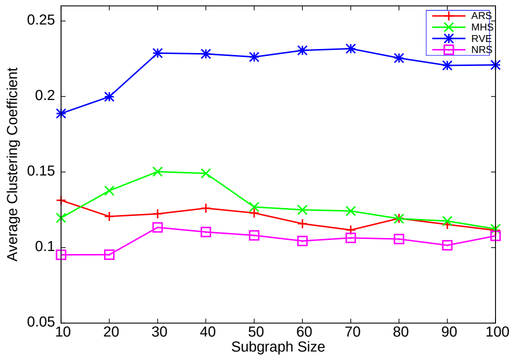


Figure 4.13: Comparison of average clustering coefficient of samples. The original Barabási-Albert graph has 500 vertices and $d = 10$.

4.3.4 Efficiency

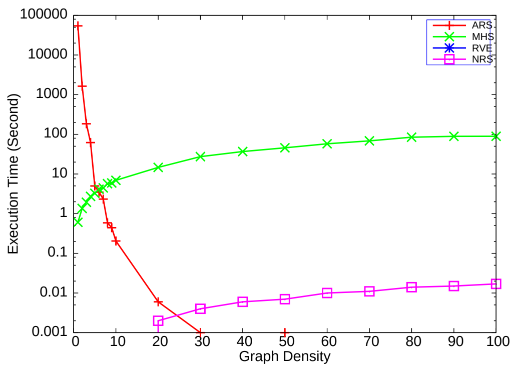
We evaluate the efficiency of the algorithms on graphs of different densities and with different subgraph sizes.

4.3.4.1 Varying Density

We measure the execution time of the four algorithms with a series of Barabási-Albert graphs with 500 vertices. The number of links that each new vertex generates for each graph varies from 1 to 10 with step 1 and from 20 to 100 with step 10. For each graph we sample 10 connected induced subgraphs of size 10 and calculate the average execution time.

The results are presented in Figures 4.14.

We see that the execution times of ARS and MHS are increasing rapidly as the graph density decreases and increases, respectively. When the graph density is below 5, ARS becomes unacceptably inefficient. However, real graphs often display a density less than 5 so that ARS is not practical in real applications. On the contrast, MHS caters for the sparse graph because of faster convergence. However, on dense graphs MHS is inefficient as the Markov Chain has numerous states so that



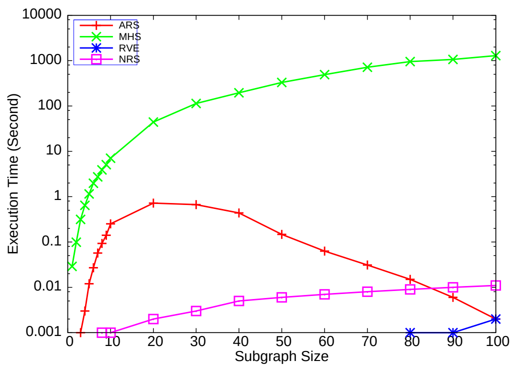


Figure 4.14: Average execution times of sampling a connected induced subgraph of size 10 from Barabási-Albert graphs with 500 vertices and different densities.

Figure 4.15: Average execution times of sampling connected induced subgraphs of different sizes from a Barabási-Albert graph with 500 vertices and $d = 10$.

the convergence is slow. Nevertheless, NRS scales well along with the graph density as its running time is bounded by the number of vertices. The execution time of NRS increases because, when the graph is denser there are more candidate vertices that can be replaced, so there are more chances to traverse a whole subgraph to confirm the connectivity. The execution time of RVE is always less than $1ms$ so that it cannot be displayed. RVE is the fastest because it terminates as long as desired number of vertices are sampled.

4.3.4.2 Varying Prescribed Size

We measure the execution time of the four algorithms with a Barabási-Albert graph with 500 vertices. The graph density is about 10. For each algorithm, the sampled subgraph size is varying from 1 to 10 with step 1 and from 20 to 100 with step 10. We compute the average execution times of 10 runs for each sample size.

The results are presented in Figures 4.15.

The execution time of ARS first increases because of the growing subgraph size, and then decreases because of fewer rejections of the samples. The execution time of MHS, as expected, is increasing when the subgraph size grows as there are more states in the Markov chain. NRS is slowly increasing when the subgraph size grows. This slow increase is because when the subgraph size increases, the test of connectivity consumes more time. RVE is still the fastest.

4.3.5 Efficiency versus Effectiveness

We measure the overall performance of the four algorithms in terms of the efficiency versus the effectiveness.

For efficiency, we compute the average execution time for each algorithm of all the settings of graph density in Figure 4.14 and all the settings of subgraph size in Figure 4.15, respectively. We normalize the execution time in $[0, 1]$. A lower value corresponds to higher efficiency.

For effectiveness, we consider ARS as the most effective algorithm because it produces a true uniform distribution. For each algorithm, we compute the average value of average degree and clustering coefficient for each setting and measure the difference from ARS by computing the sum of square errors (SSE) of all the values. We normalize the SSE in $[0, 1]$. A lower value corresponds to higher effectiveness.

The results are presented in Figure 4.16 and Figure 4.17.

We see that the effectiveness of ARS, MHS, and NRS are very close to each other. However, the efficiency of ARS and MHS depends highly on the graph density and the sampled subgraph size. RVE has the best efficiency but the worst effectiveness.

4.3.6 Sampling Graph Properties

We measure the D-statistic for six of the nine properties used in [70] for scale-down sampling (the authors of [70] also consider back-in-time sampling). The three

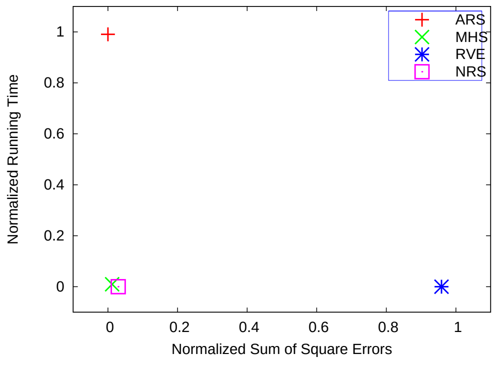


Figure 4.16: Normalized efficiency versus effectiveness of sampling connected induced subgraphs of size 10 from Barabási-Albert graphs with 500 vertices and different densities.

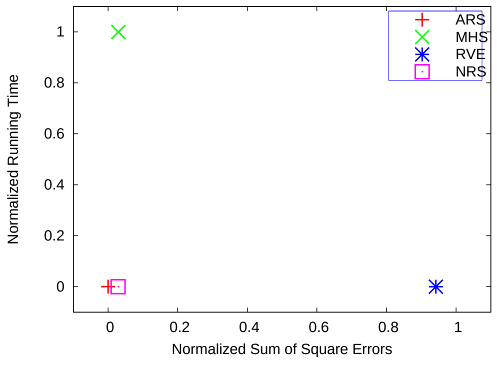


Figure 4.17: Normalized efficiency versus effectiveness of sampling connected induced subgraphs of different sizes from a Barabási-Albert graph with 500 vertices and $d = 10$.

remaining properties concern the distribution of component size and, therefore, do not apply to connected subgraph sampling.

	in-deg	out-deg	clust	sng-val	sng-vec	hops
RN	0.084	0.145	0.327	0.079	0.112	0.231
RPN	0.062	0.097	0.243	0.048	0.081	0.200
RDN	0.110	0.128	0.256	0.041	0.048	0.238
RE	0.216	0.305	0.525	0.169	0.192	0.509
RNE	0.277	0.404	0.709	0.255	0.273	0.702
HYB	0.273	0.394	0.670	0.240	0.251	0.683
RNN	0.179	*0.014*[2]	0.398	0.060	0.255	0.252
RJ	0.132	0.151	0.235	0.076	0.143	0.264
RW	0.082	0.131	0.243	0.049	0.033	0.243
FF	0.082	0.105	0.244	0.038	0.092	0.203
NRS	**0.048**	**0.074**	**0.059**	0.060	**0.012**	0.401

[2] We suspect this number is a typo.

Table 4.2: Measuring D-statistic on six graph properties.

The first three properties, the in-degree distribution, out-degree distribution, and clustering coefficient, are properties local to vertices.

The next two properties, the distribution of singular values of the graph adjacency matrix versus the rank and the distribution of the first left singular vector of the graph adjacency matrix versus the rank, are spectral properties of the graph.

The sixth property, the distribution of the number of reachable pairs of nodes at distance h or less, is a global property of the sample.

We run NRS on the same datasets and using the same experimental settings as [70], and compare the results with the corresponding ones published in [70]. The results are presented in Table 4.2. Remember that a lower value of D-statistic indicates higher similarity of two distributions.

The results suggest that sampling connected induced subgraphs uniformly at random can preserve significant properties such as degree distribution and clustering coefficient distribution. This is because connected induced subgraphs naturally contain the information of local graph properties such as vertex degree and clustering coefficient. Also, our sampling algorithm can preserve relatively well the spectral properties such as singular values and the first left singular vector of graph adjacency matrix. The overall performance of NRS is much better than the two outperforming algorithms reported in [70], FF and RW.

4.4 Discussion

ARS, MHS and NRS are effective. They sample almost uniformly at random a connected induced subgraph with a prescribed number of vertices from a graph. NRS is slightly less effective than ARS and MHS. RVE has bias towards denser subgraphs.

RVE is more efficient than the other three algorithms. NRS is practical on all graphs but slower than RVE. MHS is efficient on sparse graphs and small prescribed sizes. ARS is only efficient for very dense graphs.

The newly proposed algorithm NRS very successfully realizes the compromise between effectiveness and efficiency for which it was designed.

4.5 Summary

In this work, we study the uniform random sampling of connected induced subgraphs of a prescribed size from a graph.

We present and discuss four algorithms that leverage several ideas such as rejection sampling, random walk and Markov Chain Monte Carlo.

The first algorithm, Acceptance-rejection Sampling (ARS), provides a reference for effectiveness. It is generally not efficient.

The second algorithm, Random Vertex Expansion (RVE), illustrates the performance of a selection of vertices without replacement but has limited effectiveness.

The third algorithm, Metropolis-Hastings Sampling (MHS), demonstrates the practicality and the effectiveness of Markov Chain Monte Carlo.

The main contribution of this paper is the Neighbour Reservoir Sampling (NRS) algorithm that tries and finds a compromise between the effectiveness of a random walk on a Markov chain of connected induced subgraphs, as in MHS, with the performance of a sampling of vertices with no replacement, as in RVE.

Chapter 5

Sampling from Dynamic Graphs

5.1 Introduction

As far as we know, existing graph sampling algorithms are designed to sample from static graphs [70, 56, 75, 80]. Vertices and edges are neither added to nor removed from original graphs. However, the graphs encountered in modern applications are dynamic. Vertices are added or removed. For example, Twitter was reported to have an average of $460,000$ new users every day in February of 2011 [1]. Some of the properties of the Twitter user graph may have correspondingly changed over time.

For the sake of efficiency, at least, we want to avoid sampling the successive graphs from scratch. The challenge is to incrementally maintain a representative sample graph. We try and devise algorithms able to incrementally update the sample graph as the original graph is modified.

We consider two *undirected* graphs G and G'. We refer to G as the "old" graph and to G' as the "updated" graph. For the sake of simplicity, we consider that G' is obtained by the addition of vertices and edges to G. We do not consider the

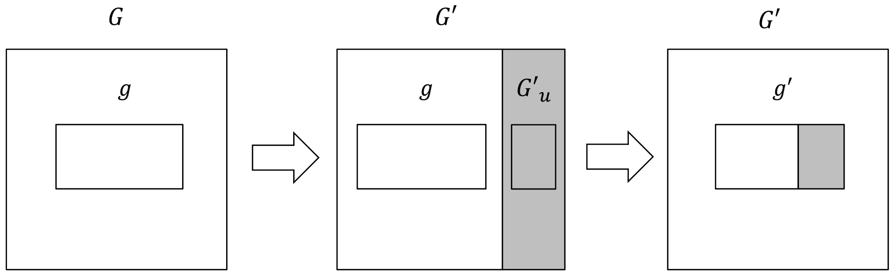


Figure 5.1: Illustration of the rationale of incremental construction of a sample g' of G' from a sample g of G. G'_u is the subgraph induced by the updated vertices of G'. g' is formed by replacing some vertices in g with the vertices of G'_u in the smaller gray rectangle.

addition of isolated vertices. We call "updated" a vertex of G' that is the endpoint of an edge in G' that does not belong to G. We consider sample graphs of fixed size. The rationale of the proposed algorithms is to replace a fraction of the vertices in the sample graph g of the graph G with updated vertices to obtain a sample graph g' of the graph G'. We refer to g as the "old" sample graph and to g' as the "updated" sample graph. Figure 5.1 illustrates this rationale.

We devise two variants of the idea above in which the vertices are replaced randomly or deterministically, respectively.

5.2 Metropolis Graph Sampling

In Section 2.3, we review the *Metropolis Graph Sampling (MGS)* algorithm proposed by Hübler et al. in [56]. The algorithm belongs to MCMC methods. The algorithm samples a subgraph g of fixed size with probability reversely proportional to its distance measure $\Delta_{G,\sigma}(g)$ to the original graph G with respect to some graph property σ. They modify the algorithm to an optimization variant by storing the subgraph with the lowest distance measure during random walk.

The MGS algorithm outperforms the *Forest Fire (FF)* algorithm and the *Ran-*

dom Walk (RW) algorithm proposed in the pioneering paper [70] with respect to both sample quality and sample size. However, MGS is not efficient because of two reasons. First, it requires a large number of random walk steps in order to make the Markov chain converge. Second, it computes the graph property σ of the subgraph and the distance measure between the subgraph and the original graph at each random walk step. We shall incrementally use the idea of MGS and reduce the number of random walk steps as well as the cost of computing σ and the distance measure. In addition, we observe (see the corresponding experimental results in Section 5.4.2.5) that MGS generates many isolated vertices in the sample graph, that is, the vertices having degree 0, especially when the sample size is small or the original graph is sparse. To avoid this problem, the *Chaining* algorithm was proposed in [56] to restrict the sample graphs to be connected. However, real graphs neither contain a large faction of isolated vertices or have only a single component. They usually consist of a few connected components of different sizes. We shall modify the MGS algorithm to generate sample graphs that are not necessarily connected and have no isolated vertex.

5.3 The Algorithms

We begin this section with a modification of the MGS algorithm for sampling from static graphs. Then we present our proposed algorithms. Both algorithms incrementally apply the modified MGS algorithm on a dynamic graph.

5.3.1 Modified Metropolis Graph Sampling

We modify the MGS algorithm to restrict the sample graphs to have no isolated vertex. To avoid the initial selection of isolated vertices, we modify the MGS algorithm in the way of selecting the initial subgraph. We randomly select a vertex in G and perform the *Breadth First Search (BFS)* graph traversal algorithm to generate the initial subgraph $g_{current}$ of size n. Alternatively, one can apply any other graph traversal algorithm to sample the initial subgraph. If we get stuck in a component of size less than n, we perform BFS with a new starting vertex, until n vertices are sampled. This method guarantees to generate an initial subgraph without any isolated vertex. To avoid the introduction of isolated vertices during random walk, we select the pair of vertices (v, w) for replacement at each step as follow. The vertex v is randomly selected from $g_{current}$. The vertex w is randomly selected from the neighbor vertices of $g_{current} \setminus \{v\}$. We form a new subgraph $g_{new} = (g_{current} \setminus \{v\}) \cup \{w\}$. We reject g_{new} directly if it contains any isolated vertex. Otherwise, we move from $g_{current}$ to g_{new} with a proposed probability. Algorithm 7 describes the pseudo code for selecting a random pair of vertices (v, w) for replacement.

Algorithm 7: `Finding_Pair`(g, G)

```
Input: $G$ : a graph, $g$ : a subgraph without isolated vertex of $G$
Output: $v, w$ : a pair of vertices for replacement.
1 $v \leftarrow$ select a vertex uniformly at random from $g$;
2 $N(g_r) \leftarrow$ find all the neighbor vertices of $g \setminus \{v\}$ in $G$;
3 $w \leftarrow$ select a vertex uniformly at random from $N(g_r)$;
4 Return $(v, w)$;
```

We thus construct a Markov chain whose states are the subgraphs without any isolated vertex of size n of G. The rest of the algorithm is as the same as the original MGS algorithm. We call this modified variant *Modified Metropolis Graph Sampling (MMGS).*

5.3.2 Incremental Metropolis Sampling

We now turn to the problem of incremental construction of a updated sample graph g' of G' from an old sample graph g of G.

One factor that determines the efficiency of the MMGS algorithm is the number of random walk steps required to achieve the stationary distribution. If we sample g' from G' using the MMGS algorithm, the number of random walk steps is relevant to the size of G'. The smaller the size of G' is, the fewer subgraphs of G' are in the Markov chain, hence the fewer random walk steps are required to achieve the stationary distribution. Therefore the efficiency of the MMGS algorithm can be improved if we can apply it to sample g' from a subgraph of G'. On the other hand, as we discussed in Section 5.1, the rationale of incremental sampling is to replace some vertices in g with the same number of updated vertices in G'. The key challenge is how to select the replaced vertices and the updated vertices. We denote by G'_{temp} the subgraph induced by the vertices in g and the updated vertices in G'. We find that the Metropolis algorithm can spontaneously solve this problem via random walk on the Markov chain whose states are the subgraphs constructed by the vertices in G'_{temp}.

In particular, we first construct the subgraph G'_{temp} and set $g_{best} = g_{current} = g$. We start the random walk from g. At each step, we randomly select a pair of vertices (v, w) for replacement using Algorithm 7, where $v \in g_{current}$ and $w \in G'_{temp} \backslash g_{current}$. We construct a new subgraph $g_{new} = (g_{current} \setminus \{v\}) \cup \{w\}$. If g_{new} contains any isolated vertex, we return to $g_{current}$. Otherwise, we accept the transition from $g_{current}$ to g_{new} with probability $\frac{\Delta_{G',\sigma}(g_{current})}{\Delta_{G',\sigma}(g_{new})}$. Reader notice that, differently from the MMGS algorithm, we construct the subgraphs $g_{current}$ and g_{new} from G'_{temp} but compute the distance measure Δ_σ between the subgraphs and the updated graph G'. In case of a success, we set $g_{current} = g_{new}$ and, if $\Delta_{G',\sigma}(g_{current}) < \Delta_{G',\sigma}(g_{best})$,

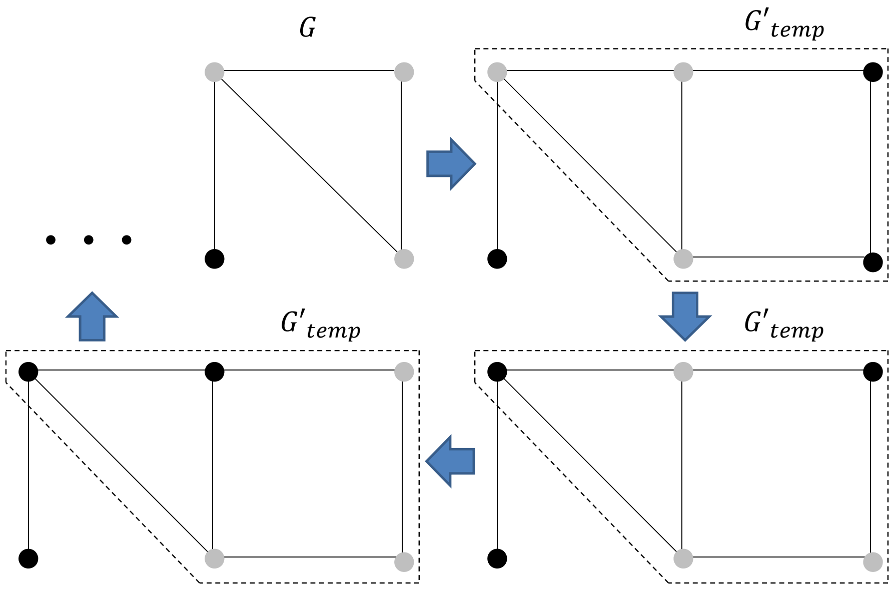


Figure 5.2: Illustration of the IMS algorithm. The subgraph in the dashed area is G'_{temp}. The size of sample graph is 3. At each step, the gray vertices construct the subgraph of the Markov chain.

we store $g_{current}$ as g_{best}. We output g_{best} as the sample graph g' of the updated graph G' after a sufficient large number of random walk steps. In summary, we sample g' from the subgraph G'_{temp} of G'. We construct a Markov chain whose states are the subgraphs without isolated vertex of size n of G'_{temp}. We perform a random walk starting from g on this Markov chain using the Metropolis algorithm. The replacement of the vertices is spontaneously realized during the random walk. We output the subgraph as g' that has the lowest distance measure to G' with respect to property σ. Figure 5.2 illustrates this algorithm with a small example. We call this algorithm *Incremental Metropolis Sampling (IMS)*. The pseudo code is described in Algorithm 8. The parameter p affects the convergence of the Markov chain. We set $p = 10 \times \frac{E}{V} \lg V$ in the experiment according to the parameter setting in [56], where E and V are the number of edges and the number of vertices of G'_{temp}, respectively.

We now turn to prove that the stationary probability of each subgraph of the

Markov chain is inversely proportional to its distance measure to the original graph. We need to prove that the Markov chain is *ergodic* and satisfies the *detailed balance*.

```
Algorithm 8: Incremental Metropolis Sampling
  Input: $g$ : a sample graph of $G$, $n$ : the sample size, $G'$ : the updated graph,
         $\#it$ : the number of random walk steps.
  Output: $g'$ : a sample graph of $G'$.
1 $G'_{temp}$ = the subgraph of $G'$ induced by the vertices in $g$ and the updated
  vertices in $G'$;
2 $g_{best} = g_{current} = g$;
3 for $i = 1$ to $\#it$ do
4     $(v, w) =$ Finding_Pair$(g_{current}, G'_{temp})$;
5     $g_{new} = (g_{current} \setminus \{v\}) \cup \{w\}$;
6     if $g_{new}$ contains any isolated vertex then
7         continue;
8     end
9     $\alpha \leftarrow$ random number from interval $[0, 1]$;
10    if $\alpha < (\frac{\Delta_{G',\sigma}(g_{current})}{\Delta_{G',\sigma}(g_{new})})^p$ then
11        $g_{current} = g_{new}$;
12        if $\Delta_{G',\sigma}(g_{current}) < \Delta_{G',\sigma}(g_{best})$ then
13            $g_{best} = g_{current}$;
14        end
15    end
16 end
17 $g' = g_{best}$;
18 Return $g'$;
```

Notice that not all the subgraphs without isolated vertex of size n of G'_{temp} can be sampled (or the Markov chain does not contain all the subgraphs without isolated vertices of size n of G'_{temp}), given the old sample graph g as the initial subgraph. Suppose g is the subgraph induced by the vertices in a set $C = \{c_1, c_2, \dots, c_k\}$ of disjoint components of G'_{temp}. Then the states of the Markov chain of the IMS algorithm can only be the subgraphs without isolated vertex of size n induced by the vertices from exactly the same set of components. On one hand, no vertex from other component can be selected during random walk. This is because we always select the vertex w for replacement from the neighbor vertices of the current subgraph. But the vertices that are not in C do no connect to any vertices in C. On the other hand, no component in C can be discarded during random walk, that is, each component in C contributes some vertices to each subgraph of the Markov chain. This is because each component in C must have at least two connected vertices being selected during random walk. Otherwise the corresponding subgraph contains isolated vertices. A possible solution for selecting vertices from all the components of G'_{temp} is to select a pair of connected vertices for replacement instead of selecting a single neighbor vertex. However, we are concerned with preserving graph properties rather than sampling from all the vertices. We prove that the Markov chain of the IMS algorithm is ergodic with respect to the subgraphs without isolated vertex of size n induced by the vertices exactly in the components of C. By exactly we mean the vertices must be in these components and each component mush contain some of the vertices. Note that the vertices discussed here are those of the subgraph G'_{temp} by default.

Lemma 1. *Given an initial subgraph g of size n whose vertices are in a set $C = \{c_1, c_2, \ldots, c_k\}$ of disjoint components of G'_{temp}, the corresponding Markov chain of the IMS algorithm is ergodic, whose states are all the subgraphs without isolated vertex of size n induced by the vertices exactly in the components of C.*

Proof. (i) Aperiodicity. It is straightforward to prove the Markov chain is aperiodic, because there are self-loops in the Markov chain by applying the Metropolis algorithm.

(ii) Irreducibility. The theorem of irreducibility states that each state of the Markov chain can be reached from any other state of the Markov chain. Suppose that g^1 and g^2 are any two subgraphs of size n without isolated vertex, induced by the vertices in the set C of components of G'_{temp}. Suppose the subgraph of

g^1 from component c_i is s_i^1, and the subgraph of g^2 from component c_i is s_i^2, for $i = 1, 2, \ldots, k$. Notice that s_i^1 and s_i^2 are subgraphs from the same component of C, for $i = 1, 2, \ldots, k$.

For each s_i^1, we can transform it to be a connected subgraph $s_i^1\prime$ by iteratively removing a current vertex and adding one neighbour vertex of the current subgraph. Therefore we can transform g^1 to $g^1\prime$ such that each $s_i^1\prime$ is connected, by transforming each s_i^1 separately, $i = 1, 2, \ldots, k$. In the same way, we can transform g^2 to $g^2\prime$ such that each $s_i^2\prime$ is connected, by transforming each s_i^2 separately. Moreover, as $s_i^1\prime$ and $s_i^2\prime$ are from the same component of C, we can find such a transformation that makes $s_i^1\prime \subseteq s_i^2\prime$ or $s_i^2\prime \subseteq s_i^1\prime$ for each i. Then the transformation from $g^1\prime$ to $g^2\prime$ are as follows. In each iteration, we find a pair of subgraphs $s_i^1\prime$ and $s_i^2\prime$ from the same component c_i of C, such that $s_i^1\prime \subseteq s_i^2\prime$. Then there must be another pair of subgraphs $s_j^1\prime$ and $s_j^2\prime$ such that $s_j^2\prime \subseteq s_j^1\prime$, $i \neq j$, as the number of vertices of g^1 and g^2 are the same. We remove one vertex from $s_j^1\prime$ and add one neighbour vertex of $s_i^1\prime$ to $s_i^1\prime$, to make the difference between $s_i^1\prime$ and $s_i^2\prime$ and the difference between $s_j^1\prime$ and $s_j^2\prime$ smaller. As all the subgraphs are connected, we can always avoid the introduction of isolated vertex. The iteration stops when $s_i^1\prime = s_i^2\prime$ or $s_j^1\prime = s_j^2\prime$. We then turn to the subgraphs from other components and start another iteration. Finally we can make sure $s_i^1\prime = s_i^2\prime$ for $i = 1, 2, \ldots, k$.

Therefore we find a path from g^1 to $g^1\prime$, then from $g^1\prime$ to $g^2\prime$ and finally from $g^2\prime$ to g^2 (by reversing the transformation from g^2 to $g^2\prime$), that is, g^2 is reachable from g^1. As g^1 and g^2 are arbitrary pair of subgraphs, the irreducibility is proved.

By taking together (i) and (ii), we proved the ergodicity of the Markov chain.

□

Now we need to prove that the detailed balance holds for the Markov chain, that is, for any two subgraphs g_1 and g_2 whose vertices fall exactly in the set C

of components of G'_{temp}, we have $\pi(g_1)p(g_1, g_2) = \pi(g_2)p(g_2, g_1)$, where $\pi(g_1)$ is the stationary probability of subgraph g_1 and $p(g_1, g_2)$ is the transition probability from g_1 to g_2.

We first prove the following lemma regarding the property of detailed balance, discussed in Section 2.1.

Lemma 2. *The detailed balance of an ergodic Markov chain holds* iff *the detailed balance holds for any two adjacent states.*

Proof. *i*) Proof of $\Rightarrow$. Suppose the detailed balance holds for the Markov chain, that is, for any two states s and $s\prime$ we have $\pi(s)p(s, s\prime) = \pi(s\prime)p(s\prime, s)$, where $\pi(s)$ is the stationary probability of s and $p(s, s\prime)$ is the transition probability from s to $s\prime$. It is obvious that the detailed balance holds for any two adjacent states.

ii) Proof of $\Leftarrow$. Suppose the detailed balance holds for any two adjacent states. For any two states s and $s\prime$ of the Markov chain, suppose there are k paths between them. We have,

$$\pi(s)p(s, s\prime) = \pi(s)\sum_{i=1}^{k} p^i(s, s\prime) = \sum_{i=1}^{k} \pi(s)p^i(s, s\prime), \tag{5.1}$$

where $p^i(s, s\prime)$ is probability of moving from s to $s\prime$ through the i^{th} path.

Denote by $Path^i(s, s\prime) =< s, s^i_1, s^i_2, \ldots, s^i_{l_i}, s\prime >$ the i^{th} path from s to $s\prime$, $i =$

$1, 2, \ldots, k$. We have,

$$\begin{aligned}
\pi(s)p^i(s, s\prime) &= \pi(s) \times p(s, s_1^i) \times p(s_1^i, s_2^i) \times \cdots \times p(s_{l_i}^i, s\prime) \\
&= p(s_1^i, s) \times \pi(s_1^i) \times p(s_1^i, s_2^i) \times \cdots \times p(s_{l_i}^i, s\prime) \\
&= p(s_1^i, s) \times p(s_2^i, s_1^i) \times \pi(s_2^i) \times \cdots \times p(s_{l_i}^i, s\prime) \\
&\cdots \\
&= p(s_1^i, s) \times p(s_2^i, s_1^i) \times \cdots \times p(s\prime, s_{l_i}^i) \times \pi(s\prime) \\
&= \pi(s\prime)p^i(s\prime, s).
\end{aligned}$$

By taking this equation into Equation 5.1, we get,

$$\pi(s)p(s, s\prime) = \sum_{i=1}^{k} \pi(s)p^i(s, s\prime) = \sum_{i=1}^{k} \pi(s\prime)p^i(s\prime, s) = \pi(s\prime)p(s\prime, s), \tag{5.2}$$

that is, the detailed balance holds for any two states of the Markov chain.

In summary, the lemma is proved by the conclusion of *i*) and *ii*). □

Lemma 3. *The detailed balance holds for the Markov chain constructed in Algorithm 8, that is, for any two subgraphs g_1 and g_2 whose vertices fall exactly in the set C of components of G'_{temp}, we have $\pi(g_1)p(g_1, g_2) = \pi(g_2)p(g_2, g_1)$.*

Proof. First, According to Lemma 2, we only need to prove that the detailed balance holds for any two adjacent subgraphs g_1 and g_2.

Second, we suppose that the stationary probability of a subgraph is reversely proportional to its distance measure to the original graph, that is,

$$\frac{\pi(g_1)}{\pi(g_2)} = \frac{\Delta_{G',\sigma}(g_2)}{\Delta_{G',\sigma}(g_1)}. \tag{5.3}$$

With the acceptance probability proposed in Algorithm 8, we get,

$$\pi(g_1)p(g_1, g_2) = \pi(g_1) \times t(g_1, g_2) \times min(1, \frac{\Delta_{G',\sigma}(g_1)}{\Delta_{G',\sigma}(g_2)})$$

and

$$\pi(g_2)p(g_2, g_1) = \pi(g_2) \times t(g_2, g_1) \times min(1, \frac{\Delta_{G',\sigma}(g_2)}{\Delta_{G',\sigma}(g_1)}),$$

where $t(g_1, g_2)$ is the probability of transition from g_1 to g_2, as discussed in Section2.1.

Without loss of generality, we assume that $\Delta_{G',\sigma}(g_1) \geq \Delta_{G',\sigma}(g_2)$. So we have,

$$\pi(g_1)p(g_1, g_2) = \pi(g_1) \times t(g_1, g_2) \tag{5.4}$$

and

$$\pi(g_2)p(g_2, g_1) = \pi(g_2) \times t(g_2, g_1) \times \frac{\Delta_{G',\sigma}(g_2)}{\Delta_{G',\sigma}(g_1)}. \tag{5.5}$$

We know that g_1 and g_2 are adjacent, that is, there is a pair of vertices (v, w), such that $g_1 \setminus \{v\} \cup \{w\} = g_2$, or $g_1 \setminus \{v\} = g_2 \setminus \{w\}$. Denote by $g_{1,2} = g_1 \setminus \{v\} = g_2 \setminus \{w\}$, which is the intermediate state between g_1 and g_2. Denote by $N(g_{1,2})$ the number of subgraphs of the Markov chain obtained by adding one vertex to $g_{1,2}$. We get,

$$t(g_1, g_2) = \frac{1}{n} \times \frac{1}{N(g_{1,2})} = t(g_2, g_1), \tag{5.6}$$

where $\frac{1}{n}$ is the probability of picking v from g_1, or picking w from g_2.

By taking together Equation 5.3, 5.4, 5.5 and 5.6, we get,

$$\begin{aligned}\pi(g_1)p(g_1, g_2) &= \pi(g_1) \times t(g_1, g_2) = \pi(g_2) \times \frac{\Delta_{G',\sigma}(g_2)}{\Delta_{G',\sigma}(g_1)} \times t(g_2, g_1) \\ &= \pi(g_2)p(g_2, g_1).\end{aligned}$$

We therefore proved that the detailed balance holds for any two adjacent states.

According to Lemma 2, the lemma holds. □

With Lemma 1, 3 and according to the theorem in Section 2.1, we can sample the subgraphs without isolated vertex using IMS with probability inversely proportional to their distance measure Δ_σ to the updated graph G' using the IMS algorithm.

5.3.3 Sample-Merging Sampling

Another factor that affects the efficiency of the MMGS algorithm is the computational cost at each random walk step. As we need to compute the neighbor vertices and the property σ of the current subgraph and the distance measure of the current subgraph to the original graph, this cost is in turn determined by the size of the subgraph. Therefore the second method to speed up the MMGS algorithm is to reduce the size of the subgraphs in the Markov chain.

To achieve this goal, we propose an algorithm based on the following conjecture. If g_x is a representative sample graph of G_x and g_y is a representative sample graph of G_y, $g_x \cup g_y$ is a representative sample graph of $G_x \cup G_y$. We assume that $G_x \cap G_y = \emptyset$. The extreme case is when $g_x = G_x$ and $g_y = G_y$. Note that $g_x \cup g_y$ is an induced subgraph of $G_x \cup G_y$. The algorithm works as follows. We construct a subgraph g_{non} that is induced by the non-updated vertices of g. We denote by G'_{non} the subgraph induced by the non-updated vertices of G'. We consider that g_{non} is a representative sample graph of G'_{non}. Suppose the size of g_{non} is s_{non}. We denote by G'_u the subgraph induced by the updated vertices of G'. We then sample a subgraph g'_u of size $n - s_{non}$ from G'_u using the MMGS algorithm. According to the conjecture, $g_{non} \cup g'_u$ is a representative sample graph of $G'_{non} \cup G'_u$, i.e., G'. We merge g_{non} and g'_u and form the sample graph g', that is, g' is the subgraph induced by the vertices in g_{non} and g'_u. In summary, we keep deterministically in

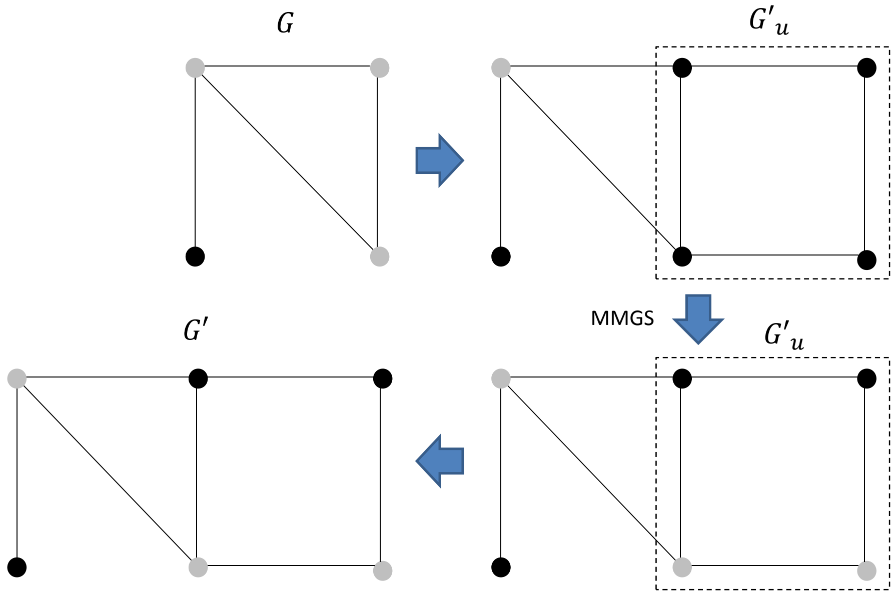


Figure 5.3: Illustration of the SMS algorithm. The subgraph is the dashed area is G'_u. The gray vertices are sampled. The size of the sample graph is 3. The size of the subgraphs of the Markov chain is 2.

g' the non-updated vertices of g and select the rest of the vertices of g' from the updated vertices of G' using the MMGS algorithm. The size of the subgraphs of the underlying Markov Chain is therefore reduced by s_{non}. By this method, we achieve the goal of reducing the size of subgraphs in the Markov chain. Figure 5.3 illustrates this algorithm with a small example. We all this algorithm *Sample-Merging Sampling (SMS)*. The pseudo code is described in Algorithm 9.

Algorithm 9: Sample-Merging Sampling

```
Input: $g$ : a sample subgraph of $G$, $n$ : the sample size, $G'$ : the updated
       graph, $\#it$ : the number of random walk steps.
Output: $g'$ : a sample graph of $G'$.
1 $g_{non}$ = the subgraph induced by the non-updated vertices of $g$;
2 $s_{non}$ = the size of $g_{non}$;
3 $G'_u$ = the subgraph induced by the updated vertices of $G'$;
4 Sample $g'_u$ of size $n - s_{non}$ from $G'_u$ using the MMGS algorithm;
5 $g' = g_{non} \cup g'_u$;
6 Return $g'$;
```

Suppose the vertices of g'_u falls in a set C of disjoint components of G'_u. It is easy to prove that the Markov chain of the SMS algorithm is ergodic with respect to the subgraphs without isolated vertex of size $n - s_{non}$ induced by the updated vertices falling exactly in C using Lemma 1 in Section 5.3.2.

The problem of SMS is that it may generates a sample graph g' with a larger distance measure to G' than g with respect to property σ, that is, $\Delta_{G',\sigma}(g') > \Delta_{G',\sigma}(g)$. The possible explanations are as follows. First, g is similar to G' with respect to property σ; secondly, g_{non} is not as a representative sample graph of G'_{non} as we expected; lastly, the merging phase introduces extra error to the distance measure between G' and g'. In this case, we restore g as a representative sample graph of G', as we are concerned with preserving graph properties rather than replacing the vertices in the old sample with the updated vertices.

5.4 Performance Evaluation

5.4.1 Complexity Analysis

We present the time complexity of the algorithms in this section. In the MMGS algorithm, we compute the neighbor vertices of a subgraph of size $n - 1$ at each step. The time complexity is $O(n)$. We also compute the graph property σ of a subgraph of size n and compute the distance measure Δ between the subgraph and the original graph. We denote by $O(n^{c_\sigma})$ the time complexity of computing property σ of a subgraph of size n. We select the Kolmogorov-Smirnov D-statistic to compute Δ. Below we see that the time complexity scales linearly with the size of the distribution (the property) in the sample graph plus that in the original graph. For the sake of simplicity, we denote by $O(n+N)$ the corrsponding time complexity, where N is the size of the original graph. Therefore the time complexity of MMGS at each step is $O(n^{c_\sigma}+2n+N)$. The number of random walk steps required is at least equal to the mixing time of the Markov chain. The mixing time of a Markov chain is theoretically studied in [100]. There are also emprical techniques for evaluating the mixing time, for instance, the *Geweke* diagnostics [39]. However, it is estimated that the number of random walk steps in the order of size of the original graph is sufficient for the convergence of our Markov chain [42, 56]. Therefore the total time complexity of MMGS is $O(n^{c_\sigma}N+2nN+N^2)$. Accordingly, the time complexity of IMS is $O(n^{c_\sigma}N_{temp}+2nN_{temp}+N_{temp}^2)$, where N_{temp} is the size of subgraph G'_{temp}. The time complexity of SMS is $O(n_u^{c_\sigma}N_u+2nN_u+N_u^2)$, where n_u is the size of subgraph g'_u and N_u is the size of subgraph G'_u.

We use the number of random walk steps discussed in this section in the experimental study below.

5.4.2 Empirical Evaluation

We conduct the experimental study in this section. We empirically evaluate the algorithms on their abilities of preserving a selected set of graph properties. All the properties are considered as distributions. We choose the Kolmogorov-Smirnov D-statistic to compute distance measure Δ_σ between the sample graph and the original graph on property σ, following the convention in [56, 70]. We run experiments on both synthetic datasets and real datasets.

5.4.2.1 The Graph Properties

We select four graph properties for evaluation, which are also used in [70].

Degree distribution: the degree distribution is defined as the probability distribution of all the degrees over a graph.

The distribution of clustering coefficient: the clustering coefficient of a vertex is defined as the ratio of existing edges between its neighbor vertices. Suppose a vertex v connects to k other vertices, where $k > 1$. The clustering coefficient of v is computed as $\frac{m}{k(k-1)/2}$, where m is the number of existing edges between these k vertices. The distribution of clustering coefficient [70] is defined as the distribution of average clustering coefficient over all vertices of the same degree.

The distribution of component size: a graph may consist of many disjoint components. The distribution of component size is defined as the distribution of the numbers of components over all the sizes. A relevant property is the *graphlet distribution* or the distribution of network motifs [61, 85]. A graphlet is a connected induced subgraph usually having 3, 4 or 5 vertices. The graphlet distribution is defined as the probability distribution of all the graphlets over a graph.

Hop-plot: $P(h)$ is defined as the number of reachable pairs of vertices at distance less than or equal to h. The hop-plot is the distribution of the numbers of vertex pairs over all the distances less than or equal to h [70].

All the properties are computed using the *Snap* [6] library.

5.4.2.2 Kolmogorov-Smirnov D-statistic

We select the Kolmogorov-Smirnov D-statistic to compute the distance measure $\Delta_{G,\sigma}(g)$ between the sample graph g and the original graph G on property σ, following the conventions in [56, 70]. The distribution (normalized) of a property of the sample graph and that of the original graph are usually have different sizes. The KS D-statistic is appropriate for this distance measure because it can measure the difference between two distributions with different sizes.

Given distribution $\mathbf{D} = [D_1, D_2, \ldots, D_{n_D}]$ and $\mathbf{d} = [d_1, d_2, \ldots$ $, d_{n_d}]$, the KS D-statistic is computed as follows. We first define a range of random variables $\mathbf{x} = [x_1, x_2, \ldots, x_k]$. Then we compute the cumulative distribution function F_D and F_d for $\mathbf{D}$ and $\mathbf{d}$ over $\mathbf{x}$, respectively, as Equation 5.7,

$$F_D(i) = \frac{1}{n_D} \sum_{j=1}^{n_D} I_j, \quad i = 1, 2, \ldots, k, \tag{5.7}$$

where $I_j = 1$ if $D_j \leq x_i$ and $I_j = 0$ otherwise.

Then the KS D- statistic is computed as,

$$D = \sup_i |F_D(i) - F_d(i)|, \quad i = 1, 2, \ldots, k. \tag{5.8}$$

A smaller value of KS D-statistic means the two distributions in comparison are more similar to each other.

5.4.2.3 Datasets

We generate Barabási-Albert random graphs [15] and Forest Fire random graphs [71] for the synthetic datasets. Both of the graph models simulate the evolution of real graphs. Barabási-Albert random graphs simulate the preferential attachment phenomenon in real graphs. Forest Fire random graphs reproduce the behavior of densification power laws and shrinking diameters in real graph evolution.

We generate 10 random graphs with different densities for each model. In the Barabási-Albert model, the density of the graph is determined by the parameter d, which is the number of edges each new vertex generates. Higher values of b lead to the generation of denser graphs. We set $b = 1, 2, \ldots, 10$ for the ten graphs, respectively. In the Forest Fire model, the density is relevant to both the forward burning probability p and the backward burning probability p_b. We set $p_b = 0.3$ and vary the value of p from 0.31 to 0.4 with increments of 0.01. Higher values of p lead to the generation of denser graphs. In order to generate multiple disjoint components in the Barabási-Albert graphs, we randomly form a new small component with some probability at the arrival of each new vertex. In the Forest Fire graphs, we generate multiple components by setting the probability of a new vertex being an orphan to be 0.1. However, we discard the orphan vertices and keep only the vertices having degree at least 1. Each synthetic graph has 10000 vertices.

The real dataset is a Facebook friendship graph. The graph contains a five-year friendship list within 15 schools with timestamp for the confirmation of each pair of friends. The graph has 331667 vertices (users) and 1391866 edges (pairs of friends). We sample 10 subgraphs of different periods from the Facebook friendship graph. Each subgraph contains 20000 vertices. The average number of edges is 24058.

5.4.2.4 Experimental Setup

We implement the Random Walk algorithm, the Forest Fire algorithm, the Modified Metropolis Graph Sampling algorithm, the Incremental Metropolis Sampling algorithm and the Sample-Merging Sampling algorithm in C++. We run all the algorithms on each of the graphs described above. We evaluate their ability of sampling each of the properties seperately. For each synthetic graph, we generate a sample graph of size 100 whenever the graph size increases by 1000. FF, RW and MMGS generate the sample graphs from scratch. IMS and SMS incrementally generate the updated sample graphs based on the old sample graphs. We compute the distance of the sample graph to the current original graph on the specific property using KS D-statistic. For each real graph, we generate a sample graph of size 200 whenever the graph size increases by 2000. We then compute the average results over the ten graphs for each graph model. We compute the average execution time of the generation of all the sample graphs for each graph model. All the experiments are run on a cluster of 54 nodes, each of which has a 2.4GHz 16-core CPU and 24 GB memory.

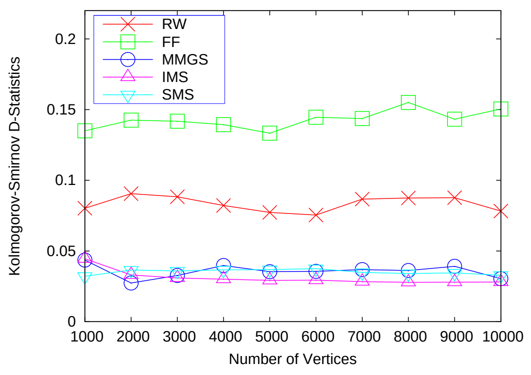


Figure 5.4: Degree distribution: Barabási-Albert graphs.

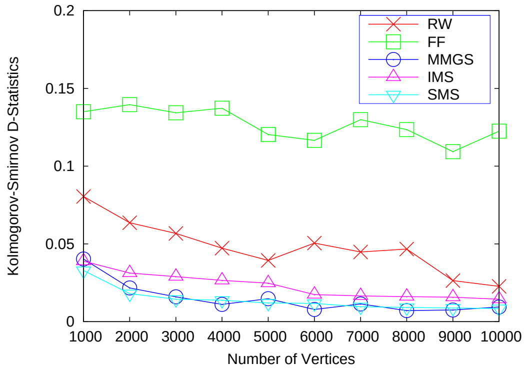


Figure 5.5: Clustering coefficient distribution: Barabási-Albert graphs.

5.4.2.5 Isolated Vertices

We observe that the original MGS algorithm generates sample graphs with a large fraction of isolated vertices. Table 5.1 shows the average fraction of the isolated vertices. Each entry represents the average fraction over all the sample graphs of the corresponding graph model for preserving the corresponding property. The results suggest that MGS generates very sparse sample graphs. Most of the vertices have no incident edge.

	Deg	clus	component	hop
Barabási-Albert	0.982	0.863	0.953	0.819
Forest Fire	0.987	0.877	0.982	0.808
Facebook Friendship	0.988	0.951	0.991	0.947

Table 5.1: The fraction of isolated vertices in the sample graphs generated by MGS.

5.4.2.6 Effectiveness

Figure 5.4, 5.5, 5.6 and 5.7 show the average results over the ten Barabási-Albert random graphs on preserving the four graph properties, respectively.

We observe that the IMS and SMS algorithms perform better than the RW and FF algorithms. They also perform similar to the MMGS algorithm. RW and FF

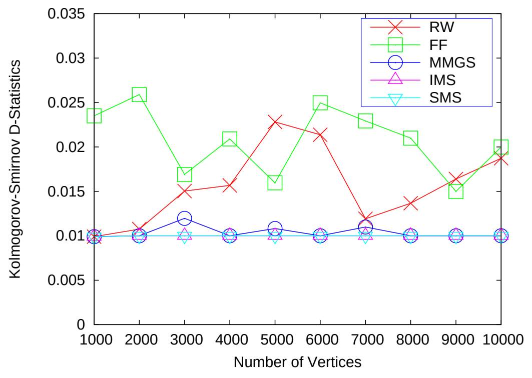


Figure 5.6: Component size distribution: Barabási-Albert graphs.

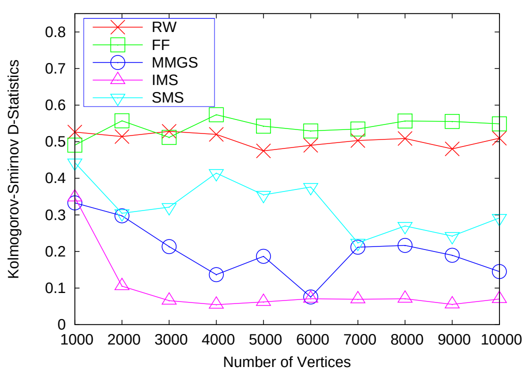


Figure 5.7: Hop-Plot: Barabási-Albert graphs.

exhibit random behaviors for the component size distribution. This is because they always tend to sample a connected subgraph from a starting vertex. The number of components in the sample graph depends highly on the selection of random starting vertex. The MMGS algorithm generally performs slightly better than the IMS and SMS algorithm. However, we observe that IMS performs better than MMGS for degree distribution and hop-plot. A possible reason is that the MMGS algorithm gets stuck in local maximums of the probability distribution of the Markov chain. Therefore it has not fully converged. The IMS algorithm roughly performs better than the SMS algorithm. This is because SMS samples local optimal subgraphs from the updated graph and introduces extra error in the merging phase.

Figure 5.8, 5.9, 5.10 and 5.11 show the average results over the ten Forest Fire random graphs on preserving the four graph properties, respectively.

We observe the similar results with the those of the Barabási-Albert graphs. The RW algorithm and the FF algorithm exhibit more random behaviors compared with the results of the Barabási-Albert graphs. MMGS, IMS and SMS perform better than RW and FF.

Figure 5.12, 5.13, 5.14 and 5.15 show the average results over the ten Facebook friendship subgraphs on preserving the four graph properties, respectively.

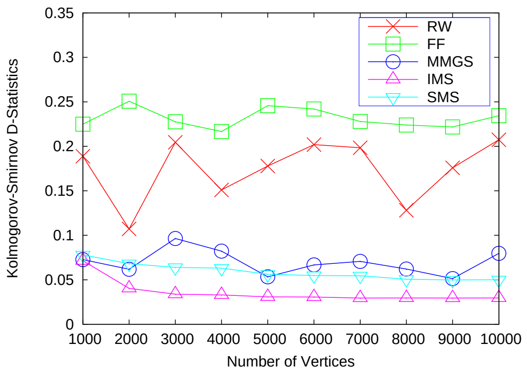


Figure 5.8: Degree distribution: Forest Fire graphs.

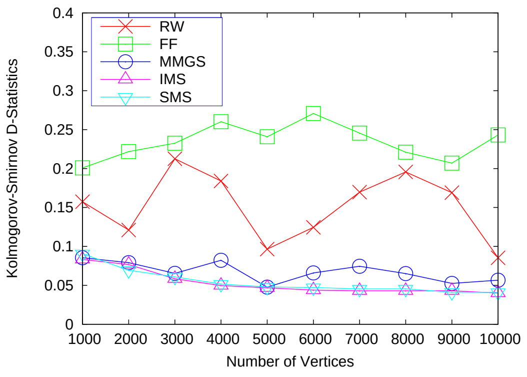


Figure 5.9: Clustering coefficient distribution: Forest Fire graphs.

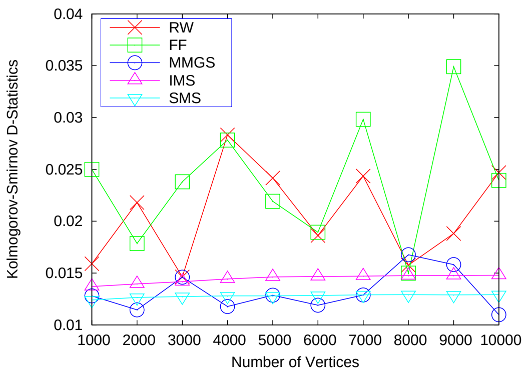


Figure 5.10: Component size distribution: Forest Fire graphs.

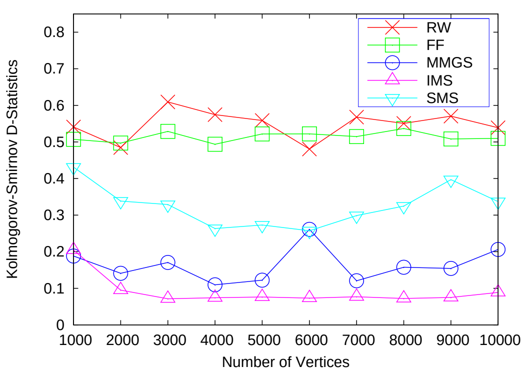


Figure 5.11: Hop-Plot: Forest Fire graphs.

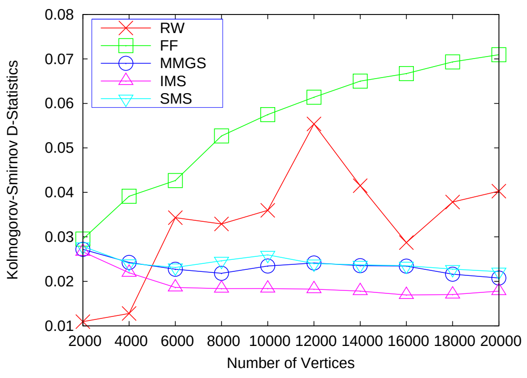


Figure 5.12: Degree distribution: Facebook friendship graphs.

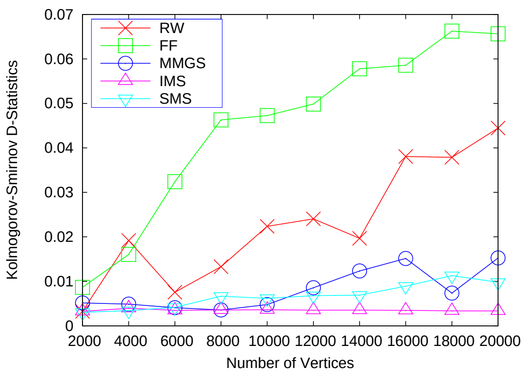


Figure 5.13: Clustering coefficient distribution: Facebook friendship graphs.

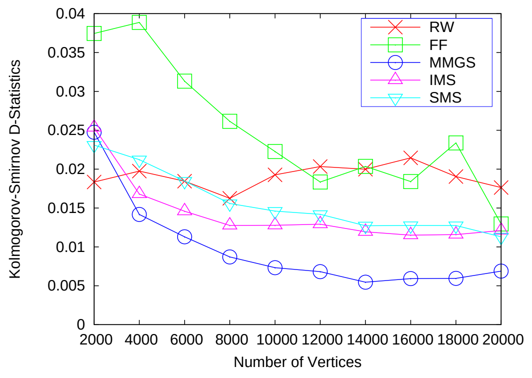


Figure 5.14: Component size distribution: Facebook friendship graphs.

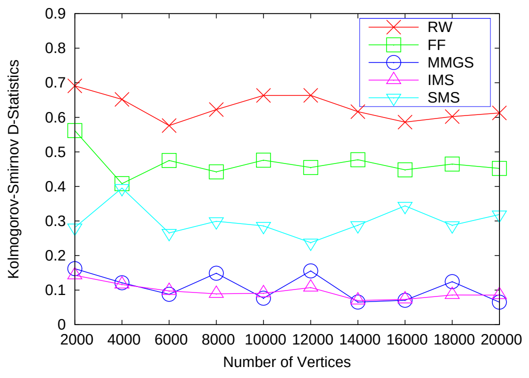


Figure 5.15: Hop-Plot: Facebook friendship graphs.

We observe that RW and FF perform similar to the other three algorithms on degree distribution, clustering coefficient distribution and component size distribution at the beginning stage of the graph evolution. To some extend, these results coincide with the finding in [70] that %15 is a good sample ratio for RW and FF to matching properties. When the size of the original graph increases, the sample ratio decreases because we fix the sample size. As a result, the quality of the sample graphs generated by RW and FF decreases accordingly. The MMGS, IMS and SMS algorithms still perform better than RW and FF.

Overall, we show that MMGS, IMS and SMS perform better than RW and FF on preserving the four properties. MMGS performs slightly better than IMS and SMS and IMS performs better than SMS in general. Among the four properties, we observe that SMS performs not good enough on preserving the hop-plot property. The possible reason is that hop-plot is a global property of the original graph, whereas SMS generates the local optimal sample graph from the updated graph. The merging phase also introduces extra error.

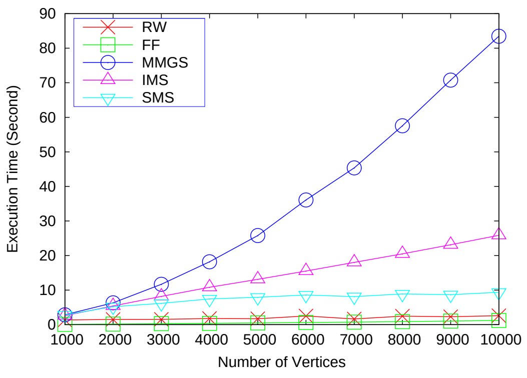


Figure 5.16: Execution Time: Barabási-Albert graphs.

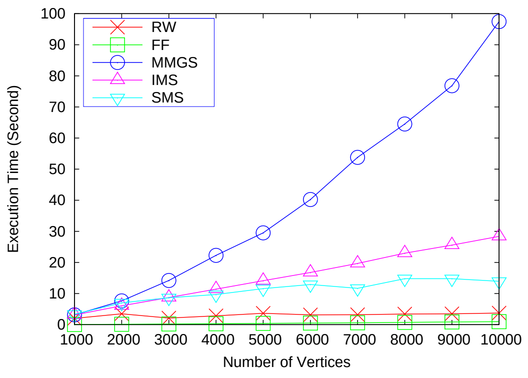


Figure 5.17: Execution Time: Forest Fire graphs.

5.4.2.7 Efficiency

Figure 5.16, 5.17 and 5.18 show the average execution time of sampling the four graph properties over the ten graphs for both synthetic and real graphs, respectively.

We observe that RW and FF are faster than the other three algorithms as expected. Our IMS and SMS algorithms are much faster than the MMGS algorithm, and SMS is faster than IMS. This is as expected because IMS and SMS require fewer random walk steps than MMGS does, and SMS computes the neighbor vertices and the graph properties of subgraphs with smaller sizes than IMS does.

The difference between the execution time of MMGS and IMS and that between MMGS and SMS increases very quickly as the original graph grows. This is because the total number of vertices of the original graph increases faster than the number of updated vertices as the original graph grows. Therefore the number of random walk steps required by MMGS increases faster than that required by IMS and SMS.

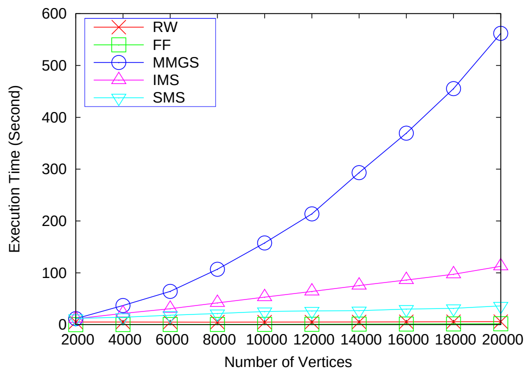


Figure 5.18: Execution Time: Facebook friendship graphs.

5.5 Summary

We have presented two incremental graph sampling algorithms preserving the distributions of degree, of clustering coefficient, of component size and of hop-plot. The rationale of the algorithms is to replace a fraction of vertices in the former sample with newly updated vertices. The two algorithms differ in their way of selecting the vertices to be replaced and updated in the sample graph.

We analytically and empirically evaluated the performance of the proposed algorithms: Incremental Metropolis Sampling and Sample-Merging Sampling. We compared their performance with that of baseline algorithms: Random Walk, Forest Fire, and Modified Metropolis Graph Sampling. The experiment results on both synthetic and real graphs showed that our proposed algorithms realize a compromise between effectiveness and efficiency.

MMGS is the most effective yet slowest of the three baseline algorithms. Our algorithms are significantly more efficient than MMGS. Although very efficient, RW and FF prove not effective. Our algorithms are less efficient than RW and FF but significantly more effective. Of the two proposed algorithms, IMS is slightly more effective than SMS, while SMS is more efficient than IMS

We have provided practical algorithms for the incremental sampling of the large dynamic graphs being created on the Internet, the World Wide Web and its numerous social media.

Chapter 6

Generating Random Graphic Sequences

6.1 Introduction

The graphs that arise from concrete applications seem to correspond to models with prescribed degree sequences. There is evidence that useful graphs in most applications domains follow a prescribed degree sequence or degree law (typically the power law). Numerous algorithms have been developed that generate graphs from prescribed degree sequences [84, 111, 38] and laws [44] or that evaluate their structure and dynamics [85]. The latter example mines and evaluates the interestingness of motifs in all kinds of graphs such as transcription networks, ecological food webs and neuron synaptic connection networks.

It is therefore necessary to provide algorithms that generate graphic sequences (realizable degree sequences), in particular uniformly at random, in order to evaluate these algorithms. Using random graphs or graphic sequence with an underlying distribution of random graphs would neglect rare but possibly significant graphic sequences.

In this study, we are interested in the random generation of graphic sequences. The problem is trivial if one wishes to generate graphic sequences according to the underlying graph distribution. The problem is particularly difficult if one wishes to generate graphic sequences uniformly at random.

We develop four algorithms for the random generation of graphic sequences [74]. Two of these algorithms that generate random graphic sequences according to the underlying distribution of random graphs are trivial and are as effective and efficient as the corresponding Erdős-Rényi algorithms. The two other algorithms generate graphic sequences uniformly at random. Our contribution is the design of the corresponding Markov Chains and the empirical evaluation of the (rapid) mixing times.

6.2 Background

Graphic sequences have received a lot of attention, both from the theoretical and practical viewpoint. Google scholar retrieves $2,500$ references for a search on graphic sequence and $6,630$ for a search on degree sequence. In this section we introduce preliminary definitions and results concerning graphic sequences as well overviews the relevant work.

6.2.1 Degree Sequence

Without loss of generality and for the sake of simplicity of exposition we consider *simple graphs*, that is undirected graphs without self-loops and multiple edges. The degree sequence of a graph is defined as follows.

Definition 2. [**Degree Sequence** [101]] *Let $G(v, e)$ be a graph with n vertices* *and m edges. The degree sequence $DS = (d_1, d_2, \ldots, d_n)$ of G is the non-increasing sequence of natural numbers corresponding to the degrees of vertices in G.*

For a variety of applications, authors have considered the realizability, construction, enumeration and, less so, counting and generation of graphs with prescribed degree sequences. The list of applications is long and increasing. For instance, the authors of [42] assess the interest of data mining results by comparing them with the results obtained from mining random graphs with the same degree sequence. Interesting patterns and rules are those who are specific to the original graph rather than those frequently appearing in graphs with the same degree sequence. The authors of [73] propose a notion of k-degree anonymity and anonymization algorithms for privacy preservation in graphs in general and in social networks in particular. In this approach, identity disclosure and its prevention depend and rely on degree sequence.

One particularly interesting problem is the random generation of graphs with prescribed degree sequences. The authors of [84] present and compare three mainstream algorithms: a naïve configuration algorithm that is painstakingly matching stubs under the prescribed sequence constraints, a local optimization algorithm and a Markov Chain Monte Carlo algorithm called the switching algorithm. Recently, the authors of [38] have proposed a polynomial time algorithm that avoids the drawbacks of backtracking and uncontrolled rejection of the three approaches above. The exact problem statement may vary depending on the nature of the graph and on additional constraints. For instance, the authors of [111] consider the uniform generation of random simple connected graphs with a prescribed degree sequence.

Other applications need to determine the structure and dynamics of graphs with prescribed degree sequences. For instance, the authors of [85] mine and evaluate the significance of motifs in transcription networks, ecological food webs and neuron synaptic connection networks. The authors of [86] investigate the number of vertices and the number of cycles in the largest component of random graphs with a given degree sequence. Connected components [26] and Hamilton cycles [29] are also investigated in the graphs with prescribed degree sequences.

6.2.2 Graphical Sequence

Therefore it is important to consider, upstream from the problems of graphs with prescribed degree sequences, the realizability, construction, enumeration, counting and generation of degree sequences themselves.

However, not every non-increasing sequence of natural numbers is a degree sequence. non-increasing sequence of natural numbers that is the degree sequence of a graph is called a *graphic sequence* or a *realizable degree sequence* as per Definition 3.

Definition 3. [**Graphic Sequence** [101]] *Let $DS = (d_1, d_2, \ldots, d_n)$ be a non-increasing sequence of natural numbers. DS is said to be a graphic sequence (or a realizable degree sequence) if and only if there exists a graph $G(v, e)$ with degree sequence DS.*

Figure 6.1 illustrates the eight possible graphs with three vertices. The eight graphs yield four different graphic sequences: (0, 0, 0), (1, 1, 0), (2, 1, 1) and (2, 2, 2). There is no graph with degree sequence (1, 0, 0), (1, 1, 1) or (2, 0, 0), for instance.

The realizability of degree sequences is first investigated by Havel [53]. Hakimi [47] then complements the work and obtains a sufficient and necessary condition for a degree sequence to be graphic. The authors of [98] show the equivalence of seven previously proposed necessary and sufficient criteria for a sequence of integers to be graphic.

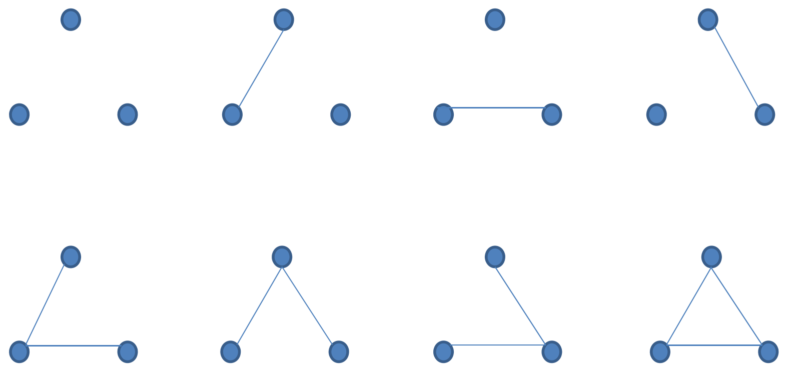


Figure 6.1: All the graphs with three vertices.

The original construction problem is trivial since a sequence of zeros is graphic. We shall use this trivial graphic sequence as a starting state in the Markov Chain Monte Carlo method that we discuss below.

As remarked by the Ruskey of [97] the enumeration of graphic sequences has been largely overlooked. They propose an algorithm that enumerates graphic sequences with prescribed length. The algorithm leverages Havel and Hakimi's condition [53]. The authors conjecture and verify experimentally that their algorithm is running in constant amortized time, i.e. in time proportional to the size of the output. Nevertheless its worst case complexity remains exponential. (see [24] and the following discussion about counting.)

Barnes and Savage propose in [17] an algorithm for the enumeration of graphic sequences with prescribed sum.

There is no known analytical formula for the counting of graphic sequences with prescribed length. Burns [24] gives a upper bound of $4^n/(\log n)^C\sqrt{n}$ and a lower bound of $4^n/Cn$.

Other authors have addressed related but different problems. Barnes, in [16], proposes a recurrence and a polynomial-time algorithm for counting graphic sequences with prescribed sum. Stanley, in [103] and Peled, in [93], count the number of ordered graphic sequences (as opposed to graphic sequences which are multi-sets) as a (exponential) function of the number of odd cycles in the corresponding forest.

While the construction problem, without or with further constraints, is rather simple, the counting problem seems difficult. This apparent paradox makes the problem of counting, and the problem of generating graphic sequences uniformly at random, interesting and challenging. We therefore are interested in addressing the random generation of graphic sequences with prescribed length and with prescribed length and sum.

6.3 The Algorithms

In this section, we present four algorithms for random generation of graphic sequences. The first two algorithms generate random graphic sequences with prescribed length, and prescribed length and sum, according to the underlying distribution of graphs. They are naïve and trivial. We give them for reference and completeness. The third and fourth algorithms generate uniformly at random graphic sequences with prescribed length, and prescribed length and sum. We contribute the first non trivial algorithms for these tasks and study their effectiveness and efficiency.

6.3.1 Random Graphic Sequence with Prescribed Length

We consider random graphic sequences with prescribed length according to the underlying distribution of the corresponding graphs which is defined by the $\Gamma_{n,p}$ Erdős-Rényi model for some probability p. The underlying graphs have a prescribed number of vertices which corresponds to the length of the sequence. We call this model $\mathcal{D}(n, p)$.

We propose a straightforward algorithm $D(n,p)$ for $\mathcal{D}(n,p)$, which is shown in Algorithm 10. We first generate a random graph with prescribed number of vertices using algorithms in [90] for the $\Gamma_{n,p}$ Erdős-Rényi model [33, 40]. Then we output its degree sequence. It is possible, as a minor optimization, to make the economy of the generation of the graph and modify a random graph generation algorithm to incrementally maintain the degree sequence. The algorithm is correct by construction.

Algorithm 10: Algorithm $D(n,p)$

```
Input: $n$ : the length of the sequence; $p$ : the probability to add an edge
Output: $DS$ : a graphic sequence generated from underlying distribution of
        $\mathcal{D}(n,p)$
1 $G = \emptyset$;
2 for $i = 1$ to $n(n-1)/2$ do
3     Generate a uniform random number $\alpha \in [0,1)$;
4     if $\alpha < p$ then
5         $G \leftarrow e_i$;
6     end
7 end
8 $DS$ = the degree sequence of $G$;
```

6.3.2 Random Graphic Sequence with Prescribed Length and Sum

We consider random graphic sequences with prescribed length and sum according to the underlying distribution of the corresponding graphs which is defined by the $\Gamma_{n,m}$ Erdős-Rényi model. The underlying graphs have a prescribed number of vertices, n, which corresponds to the length of the sequence, and a prescribed number of edges, m, which corresponds to half of the sum s of the sequence (by the Handshaking Lemma.) We call this model $\mathcal{D}(n,s)$.

One possible algorithm, which we call $D(n,s)$, which is incremental and can therefore be used to generate series of random results for increasing sums, consists in adapting the Reservoir sampling algorithm [112] to a random generation of $m = \frac{s}{2}$ edges among $\frac{n\times(n-1)}{2}$ possible ones (for a simple graph). The algorithm is shown in Algorithm 11.

Algorithm 11: Algorithm $D(n, s)$

```
Input: $n$ : the length of the sequence; $s$ : the sum of the sequence
Output: $DS$ : a graphic sequence generated from underlying distribution of
        $\mathcal{D}(n, s)$
1  $G = \emptyset$;
2  for $i = 1$ to $s/2$ do
3      $G_{e_i} \leftarrow e_i$;
4  end
5  for $i = s/2 + 1$ to $n(n-1)/2$ do
6      Generate a uniform random number $\alpha \in [0, 1)$;
7      $\mu = floor(\alpha \times i)$;
8      if $\mu \leq s/2$ then
9          $G_{e_\mu} = e_i$;
10     end
11 end
12 $DS$ = the degree sequence of $G$;
```

6.3.3 Uniformly Random Graphic Sequence with Prescribed Length

We consider graphic sequences with prescribed length uniformly at random. We call this model $\mathcal{D}_u(n)$.

Here a naïve algorithm to generate sequences in the model $\mathcal{D}_u(n)$ consists in enumerating the different graphic sequences (for instance using the algorithm of [97]) and then choosing one at random among the different ones. This approach is running in exponential time.

A Markov chain Monte Carlo approach may be able to give us acceptable approximations (almost uniform) algorithms in polynomial time. The two issues at hand are the construction of the Markov chain and the evaluation of its mixing time.

In order to illustrate the principles and the difficulties underlying the design of the Markov chain algorithm for $\mathcal{D}_u(n)$, allow us to present the construction of a simple Markov chain that fails to achieve this goal. Conveniently[1], we start from a sequence of zeros which is a graphic sequence. At each transition, we increment or decrement by 1 two elements in the sequence (which corresponds to adding or deleting an edge in the underlying graph). Each state corresponds to a sequence[2]. We only consider the new state if the elements of its corresponding sequence are in the $[0, n-1]$ interval. We only consider the new state if it is a sequence in non-increasing order. We only consider the new state if it is a graphic sequences. This is test using one of the available necessary and sufficient conditions. We thus define a Markov chain whose states are exactly the graphic sequences of prescribed length. The Markov chain is ergodic and has a stationary distribution. Unfortunately different states have different stationary probabilities. It is possible to modify the weight to make the stationary distribution uniform but it is difficult to find an a priori strategy to do so while randomly walking and constructing the Markov chain.

We can now present the construction of the Markov chain for the algorithm $D_u(n)$ that we advocate. We consider a chain that contains both graphic and non-graphic sequences. The initial state is still, conveniently, the zero sequence as it is graphic. At each transition, we increment or decrement by 1 one element in the sequence. We only consider a new state if its elements are in the $[0, n-1]$ interval. We only consider the new state if it is in non-increasing order. We do not consider states that are non-graphic and whose sum is even. However the Markov chain

[1]The initial state could be any other state.

[2]We use the words “state”and “sequence” indistinctively.

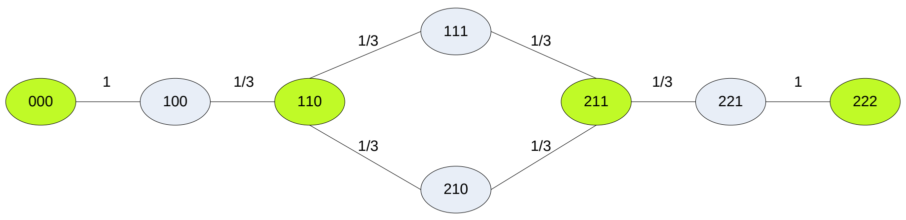


Figure 6.2: Markov chain for $n = 3$. The green ovals are graphic sequences and the gray ovals are non-graphic sequences. The weights associated to the edges lead to the uniform stationary distribution.

contains both graphic and non-graphic sequences. We call this Markov chain MC, We call P its transition matrix.

Figure 6.2 illustrates the Markov chain for three vertices, MC, in which $D_u(n)$ is walking. The green ovals represent the graphic sequences and the gray ovals represent the non-graphic sequences.

We consider the Markov chain MC' with transition matrix P^2. We cannot directly construct and walk on MC'. Rather we will walk on MC and consider even numbers of steps.

We remark that two adjacent states in MC cannot be both graphic or both non-graphic. Therefore MC has the interesting property that states that correspond to graphic sequences can only be reached in an even number of steps (from a graphic state). This may look as if compromised the aperiodicity of the Markov chain but it does not as far as we are concerned since we will walk even numbers of steps and are only interested in graphic states.

Furthermore, because we have not included in MC the non-graphic states whose sum is even, only graphic states can be reached in an even number of steps (from a graphic state). Therefore MC' is irreducible and aperiodic, therefore ergodic, as shown below with Lemma 4 and Lemma 5. The proof of Lemma 4 also implies that all the graphic states are included in our Markov chain.

Lemma 4. *MC' is* irreducible*, that is, each graphic state can be reached from any* *other graphic state in the Markov chain.*

Proof. Given two arbitrary graphic states S_i and S_j in MC, we can always find a path between them as follows. Denote by G_i and G_j the corresponding possible graphs of S_i and S_j. Consider the following procedure. We first delete the edges of G_i one by one until G_i has no edge, and then add edges one by one to construct G_j. In this procedure, each adding or deleting has a corresponding two-step path in the Markov chain MC between two graphic states. Then a path from S_i to S_j is simply a connection of all these two steps. Thus, we prove the irreducibility MC'. $\square$

Lemma 5. *MC' is* aperiodic.

Proof. Consider any two steps of random walk from a graphic state S_i in MC. If another graphic state S_j is visited, the two edges from S_i to S_j can be considered as one edge in MC'. If we walk back to S_i, these two edges indeed constitute a self-loop in MC'. As each graphic state in MC' has at least one such self-loop, MC' is aperiodic. $\square$

Unfortunately different states have different stationary probabilities. It is possible to modify the weight to make the stationary distribution uniform. We use a technique suggested by Sinclair in [100] and used by the authors of [50], that consists in allocating appropriate weights to the transitions. The result that authorizes and justifies these changes is given by Cover and Thomas in [30]. Namely the stationary probability of each state is proportional to the sum of the weights of its incident transitions, with the following transition matrix P.

$$p_{(i,j)} = \begin{cases} \frac{w_(i,j)}{\sum_{l \in adj(i)} w_(i,l)} & \text{if } j \in adj(i) \\ 0 & \text{if } j \notin adj(i), \end{cases}$$

where $w_(i,j)$ is the weight of edge corresponding to the transition from state i to j.

Lemma 6. *In MC', the stationary probability for a graphic state is proportional to the total weight incident to that state and the stationary probability for a non-graphic state is zero for the transition Matrix P^2.*

Proof. As in MC' the graphic states have even sums and the non-graphic states have odd sums, we denote by π_{e_i} the stationary probability for graphic state s_{e_i} and π_{o_i} the stationary probability for non-graphic state s_{o_i}, and denote by w_{e_i} the total weight for s_{e_i} and W the sum of total weights for all s_{e_i}. Given the distribution $\pi_e = (\pi_{e_1}, \pi_{e_2}, \ldots, \pi_{e_k}, \pi_{o_1}, \pi_{o_2}, \ldots, \pi_{o_l})$, such that

$$\begin{aligned} \pi_{e_i} &= \frac{w_{e_i}}{W}, \quad i = 1, 2, \ldots, k, \\ \pi_{o_i} &= 0, \qquad i = 1, 2, \ldots, l. \end{aligned}$$

we prove that $\pi_e = \pi_e P^2$.

Denote by $\pi_e^{(1)} = \pi_e P$ and $\pi_e^{(2)} = \pi_e P^2$, we have

$$\begin{aligned} \pi_{e_i}^{(1)} &= \sum_{j=1}^{k} \pi_{e_j} p(e_j, e_i) + \sum_{j=1}^{l} \pi_{o_j} p(o_j, e_i), \quad i = 1, 2, \ldots, k, \\ \pi_{o_i}^{(1)} &= \sum_{j=1}^{k} \pi_{e_j} p(e_j, o_i) + \sum_{j=1}^{l} \pi_{o_j} p(o_j, o_i), \quad i = 1, 2, \ldots, l. \end{aligned}$$

Because $p(e_j, e_i) = 0$, $p(o_j, o_i) = 0$ and $\pi_{o_j} = 0$, we get

$$\begin{aligned}
\pi_{e_i}^{(1)} &= 0, \quad i = 1, 2, \ldots, k, \\
\pi_{o_i}^{(1)} &= \sum_{e_j \in adj(o_i)} \pi_{e_j} p(e_j, o_i) = \sum_{e_j \in adj(o_i)} \frac{w_{e_j}}{W} \times \frac{w(e_j, o_i)}{w_{e_j}} \\
&= \frac{\sum_{e_j \in adj(o_i)} w(e_j, o_i)}{W}, \quad i = 1, 2, \ldots, l.
\end{aligned}$$

Similarly, We have

$$\begin{aligned}
\pi_{e_i}^{(2)} &= \sum_{j=1}^{k} \pi_{e_j}^{(1)} p(e_j, e_i) + \sum_{j=1}^{l} \pi_{o_j}^{(1)} p(o_j, e_i), \quad i = 1, 2, \ldots, k, \\
\pi_{o_i}^{(2)} &= \sum_{j=1}^{k} \pi_{e_j}^{(1)} p(e_j, o_i) + \sum_{j=1}^{l} \pi_{o_j}^{(1)} p(o_j, o_i), \quad i = 1, 2, \ldots, l.
\end{aligned}$$

As $p(e_j, e_i) = p(o_j, o_i) = \pi_{e_j}^{(1)} = 0$, we get

$$\begin{aligned}
\pi_{e_i}^{(2)} &= \sum_{o_j \in adj(e_i)} \pi_{o_j}^{(1)} p(o_j, e_i) = \sum_{o_j \in adj(e_i)} \frac{\sum_{e_x \in adj(o_j)} w(e_x, o_j)}{W} \times \frac{w(o_j, e_i)}{\sum_{e_x \in adj(o_j)} w(o_j, e_x)} \\
&= \frac{\sum_{o_j \in adj(e_i)} w(o_j, e_i)}{W} = \frac{w_{e_i}}{W}, \quad i = 1, 2, \ldots, k, \\
\pi_{o_i}^{(2)} &= 0, \quad i = 1, 2, \ldots, l.
\end{aligned}$$

Here we use the fact that $w(i, j) = w(j, i)$. Thus, $\pi_e = \pi_e P^2$ and π_e is the stationary distribution of MC'. $\square$

For state i and j where i and j are adjacent, the weight $w(i, j)$ is assigned as per Equation 6.1 where d_i is the degree of state i. Remember that i and j cannot be both graphic or both non-graphic.

$$w(i,j) = \begin{cases} \frac{1}{d_i} & \text{if } i \text{ is a graphic state and } j \text{ is a non-graphic state} \\ \frac{1}{d_j} & \text{if } i \text{ is a non-graphic state and } j \text{ is a graphic state.} \end{cases} \tag{6.1}$$

Note that $w(i,j) = w(j,i)$. Consequently, the sum of weights of every graphic state is 1. According to Lemma 6, the stationary distribution of graphic states is uniform. Figure 6.2 shows the example of assigning the weights to the edges of the Markov chain.

In summary, $D_u(n)$ is an algorithm that builds and randomly walks in the Markov chain MC with transition matrix P, yet outputs only evenly (or oddly, depending on the initial state) reachable states. It simulates the ergodic Markov chain MC' with transition matrix P^2 which has a uniform stationary distribution thanks to a modification of the weights in MC. The algorithm is illustrated in Algorithm 12. The parity of t depends on the initial state.

Algorithm 12: Algorithm $D_u(n)$

```
Input: $n$ : the length of the sequence; $t$ : the steps of random walk
Output: $DS$ : a graphic sequence generated from uniform distribution
1 Start from any initial state $S_0$ of non-increasing sequence;
2 $i = 1$;
3 while $i \leq t$ do
4     Compute locally the transition matrix $P$;
5     Transfer from state $S_{i-1}$ to $S_i$ according to $P$;
6     $i = i + 1$;
7 end
8 $DS = S_t$;
```

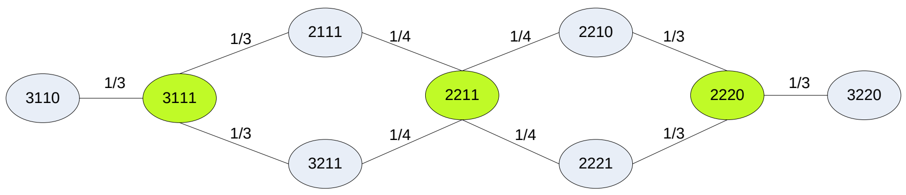


Figure 6.3: Markov chain for $n = 4, s = 6$. The green ovals are the graphic sequences and the gray ovals are the non-graphic sequences.

6.3.4 Uniformly Random Graphic Sequence with Prescribed Length and Sum

We consider random graphic sequences with prescribed length and sum uniformly at random. We call this model $\mathcal{D}_u(n, s)$.

We propose an algorithm, which we call $D_u(n, s)$. It is also a Markov chain Monte Carlo algorithm. The Markov chain in which $D_u(n, s)$ is walking is similar to the one in which $D_u(n)$ is walking. However $D_u(n, s)$ considers less states than $D_u(n)$ since the graphic sequences of the model $\mathcal{D}_u(n, s)$ are only those with prescribed sum s. The states in the Markov chain in which $D_u(n, s)$ is walking are restricted to those that correspond to sequences with sum s , for graphic states and $s - 1$ and $s + 1$, for non-graphic states. Figure 6.3 illustrates the Markov chain in which $D_u(n, s)$ is walking for $n = 4$ and $s = 6$. The weights are assigned using the method discussed in Section 6.3.3. We do not illustrate the algorithm as it is a simple variation of Algorithm 12.

6.4 Practical Optimization for $D_u(n)$

A practical optimization for $D_u(n)$ is based on the observation that complement graphs have related graphic sequences. We define the *complement sequence* to a graphic sequence as follows.

Definition 4. Complement Sequence *The complement sequence of a graphic sequence* $(d_1, d_2, \dots, d_n)$ *is* $(n-1-d_n, n-1-d_{n-1}, \dots, n-1-d_1)$.

Lemma 7. *The complement sequence of a graphic sequence is graphic.*

Before proving this lemma, we introduce a known theorem that is useful in our proof.

Theorem 1. (Erdös-Gallai) [32] *Let* $D = (d_1, d_2, \dots, d_n)$ *be a sequence of integers with* $n > d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. *Then* D *is a graphic sequence if and only if,* $\sum_i d_i$ *is even and for* $i = 1, 2, \dots, n-1$,

$$\sum_{j=1}^{i} d_j \leq i(i-1) + \sum_{j=i+1}^{n} min\{i, d_j\}.$$

Proof. Suppose $D = (d_1, d_2, \dots, d_n)$ is a graphic sequence. $\sum_i d_i$ is even according to Theorem 1. Let $d'_i = n-1-d_{n+1-i}$. The complement sequence $\{n-1-d_n, n-1-d_{n-1}, \dots, n-1-d_1\}$ is then equal to $\{d'_1, d'_2, \dots, d'_n\}$. Apparently, $\sum_i d'_i = \sum_i (n-1-d_i) = n(n-1) - \sum_i d_i$ is even. We then try to prove that for $i = 1, 2, \dots, n-1$,

$$\sum_{j=1}^{i} d'_j \leq i(i-1) + \sum_{j=i+1}^{n} min\{i, d'_j\}, \tag{6.2}$$

according to Theorem 1.

Let $left = \sum_{j=1}^{i} d'_j = \sum_{j=n-i+1}^{n} (n-1-d_j) = i(n-1) - \sum_{j=n-i+1}^{n} d_j$ and $right = i(i-1) + \sum_{j=i+1}^{n} min\{i, d'_j\}$. Assume that $i < d'_j$ for $1 \leq j \leq k$ and $i \geq d'_j$ for $k+1 \leq j \leq n$. We analyze $right$ in the following three cases:

(a.) $1 \leq k \leq i$, then

$$right = i(i-1) + \sum_{j=i+1}^{n} d'_j = i(i-1) + \sum_{j=1}^{n-i} (n-1-d_j) = i(i-1) + (n-i)(n-1) - \sum_{j=1}^{n-j} d_j.$$

According to Theorem 1 and the condition that $\{d_1, d_2, \dots, d_n\}$ is graphic, we have

$$\sum_{j=1}^{n-j} d_j \leq (n-i)(n-i-1) + \sum_{j=n-i+1}^{n} min\{n-i, d_j\} \leq (n-i)(n-i-1) + \sum_{j=n-i+1}^{n} d_j.$$

Thus,

$$\begin{aligned} right &\geq i(i-1) + (n-i)(n-1) - (n-i)(n-i-1) - \sum_{j=n-i+1}^{n} d_j \\ &= i(n-1) - \sum_{j=n-i+1}^{n} d_j = left. \end{aligned}$$

(b.) $i+1 \leq k \leq n-1$, then

$$\begin{aligned} right &= i(i-1) + \sum_{j=i+1}^{k} i + \sum_{j=k+1}^{n} d'_j = i(i-1) + i(k-i) + \sum_{j=1}^{n-k}(n-1-d_j) \\ &= i(k-1) + (n-k)(n-1) - \sum_{j=1}^{n-k} d_j. \end{aligned}$$

Still using Theorem 1, we have

$$\begin{aligned} \sum_{j=1}^{n-k} d_j &\leq (n-k)(n-k-1) + \sum_{j=n-k+1}^{n} min\{n-k, d_j\} \\ &\leq (n-k)(n-k-1) + \sum_{j=n-k+1}^{n-i} (n-k) + \sum_{j=n-i+1}^{n} d_j \\ &= (n-k)(n-i-1) + \sum_{j=n-i+1}^{n} d_j. \end{aligned}$$

Thus, we get

$$\begin{aligned} right &\geq i(k-1) + (n-k)(n-1) - (n-k)(n-i-1) - \sum_{j=n-i+1}^{n} d_j \\ &= i(n-1) - \sum_{j=n-i+1}^{n} d_j = left. \end{aligned}$$

(c.) $k = n$, then

$$right = i(i-1) + i(n-i) = i(n-1) \geq left.$$

So we proved Equation 6.2 and that the complement sequence $\{n-1-d_n, n-1-d_{n-1}, \ldots, n-1-d_1\}$ is graphic. $\square$

The practical optimization is therefore as follows. The number of states that Algorithm 12 visits can be reduced to half. The Markov chain only includes states that correspond to sequences with sum of degrees less than or equal to $n(n-1)/2$. For every random sequence generated, we then further toss a coin to decide whether we output the sequence or its complement (or, keep the sequence or discard it for those sequences who equal their complement.)

6.5 Performance Evaluation

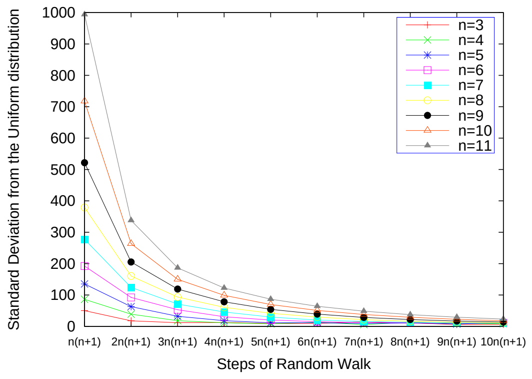
In this section, we empirically evaluate the effectiveness and efficiency of the proposed algorithms. We implement the algorithms and run the experiments on a Microsoft Windows 7 machine with an Intel Core 2 Quad 2.83G CPU and 3GB memory. All the algorithms are implemented using Visual C++ 9.0.

We do not need to evaluate the effectiveness of $D(n, p)$ and $D(n, s)$ as it derives from the effectiveness of the algorithms that we use for the generation of graphs in the Erdős-Rényi model, respectively. Their efficiency is the same as the one of the underlying Erdős-Rényi algorithm.

The effectiveness and efficiency of $D_u(n)$ and $D_u(n, s)$ are evaluated by measuring the fitness of a sample generated by the algorithm to the uniform distribution after each transition. We use for that purpose the standard deviation. We have verified and confirmed the results and the conclusions below with other tests of fitness such as χ^2 and Jensen-Shannon divergence. We evaluate the efficiency, that is the number of steps, needed to achieve a desired effectiveness, measured by the test of fitness.

6.5.1 A Lower Bound for $D_u(n)$ Mixing Time

We can now give an analytical lower bound for the mixing time of $D_u(n)$. We can show that the diameter of the Markov chain MC is $n(n-1)$. For any two graphic states i with sum s_i and j with sum s_j, there is a path from i to j whose length is $s_i + s_j$ via the sequence of all zeros and another path from i to j whose length is $2n(n-1)-(s_i+s_j)$ via the sequence of all $(n-1)$s. So no matter $s_i+s_j \leq n(n-1)$ or not, there always exists a path between any two graphic states whose length is less than or equal to $n(n-1)$. As the distance between the sequence of all zeros and the sequence of all $(n-1)$s is exactly $n(n-1)$, the diameter (the longest shortest path) of the Markov chain is $n(n-1)$. Since [72] states that the mixing time for any error $\varepsilon < 1/2$ of a Markov chain is larger than or equal to half its diameter, a lower bound for the mixing time of MC is $\frac{n(n-1)}{2}$.



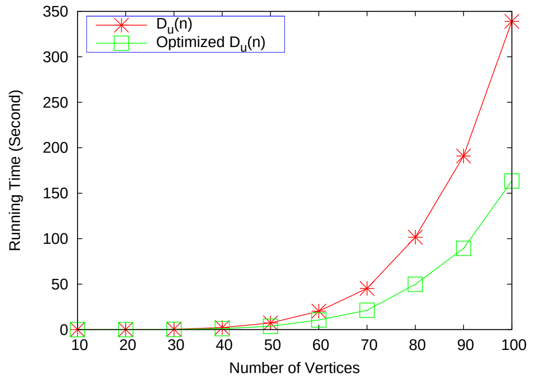


Figure 6.4: Standard Deviation from Uniform for varying number of steps for $D_u(n)$ for different n.

Figure 6.5: Running time of $D_u(n)$ and $D_u(n)$ with the practical optimization.

6.5.2 Performance of $D_u(n)$

We vary the length n of the sequences from 3 to 11 for $D_u(n)$. We measure the standard deviation for number of steps in MC varying from $n \times (n+1)$ to $10 \times n \times (n+1)$ (there are half this number of steps in MC') in increments of $n \times (n+1)$.

Figure 6.4 illustrates the standard deviation for varying number of steps for $D_u(n)$. For each n the number of generated graphic sequences is $k \times d(n)$ where $d(n)$ is the number of distinct graphic sequences of length n and $k = 100$ in the experiments. The numbers of distinct graphic sequences of some lengths can be found in A004251 of Sloane and Plouffe's encyclopedia [102].

We observe from Figure 6.4 that the algorithm is quickly reaching a minimum standard deviation (which is not 0 because of the parameter k). We can also see that this empirical mixing time increases with n. Empirically, we conservatively estimate the mixing time to be n^3 number of steps. Figure 6.5 shows the average running time for the generation of 10 graphic sequences of $D_u(n)$ for n varying from 10 to 100 in increments of 10. The curve is expected to be of the order of n^3 by construction. It however reveals the actual value of a high constant which is the cost of one step in the Markov chain. We further notice that this constant

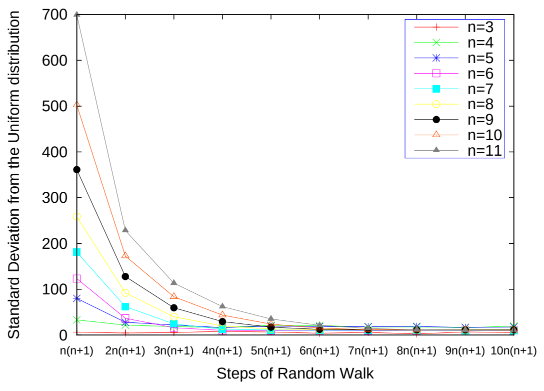


Figure 6.6: Standard Deviation from Uniform for varying number of steps for $D_u(n)$ with the practical optimization for different n.

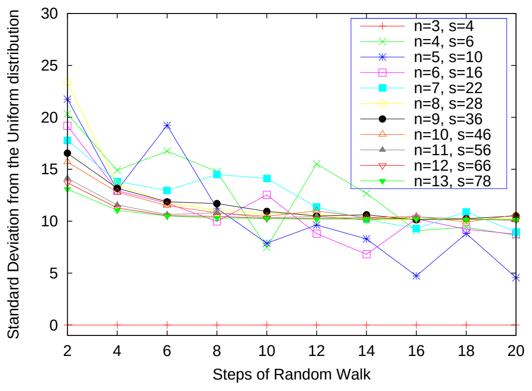


Figure 6.7: Standard deviation from Uniform for varying number of steps for $D_u(n, s)$ for different n and $s = 2 \times \lceil \frac{n\times(n-1)}{4} \rceil$.

depends on n. Therefore the actual complexity of the algorithm should combine the number of steps with the processing at each step. We can generate a graphic sequence of 100 elements uniformly at random in 340 seconds.

The mixing time and the running time can be effectively divided by 2 using the practical optimization that we have proposed. Figure 6.6 and Figure 6.5 show the mixing and running performance of $D_u(n)$ with the practical optimization. We observe significant speedup of convergence of the Markov chain. We can generate a graphic sequence of 100 elements uniformly at random in 160 seconds.

6.5.3 Performance of $D_u(n, s)$

We vary the length n of the sequences from 3 to 13 for $D_u(n, s)$. We fix the value of s to $2 \times \lceil \frac{n\times(n-1)}{4} \rceil$. With this value of s, the number of distinct graphic sequences is the largest for length n. We measure the standard deviation for number of steps in MC varying from 2 to 20 in increments of 2.

Figure 6.7 illustrates the standard deviation for varying number of steps for $D_u(n, s)$. For each n the number of generated graphic sequences is $k \times d(n, s)$ ($k =$

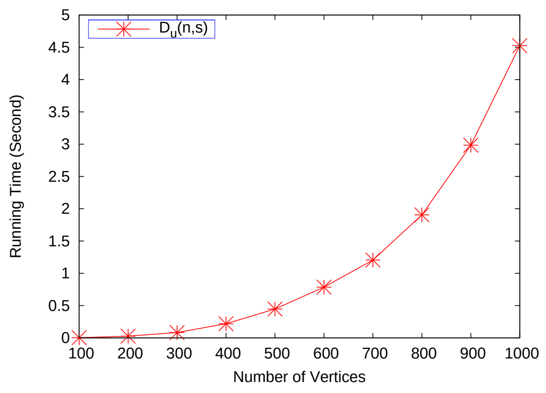


Figure 6.8: Running time of $D_u(n, s)$, for n varies in $\{100, 200, \ldots, 1000\}$.

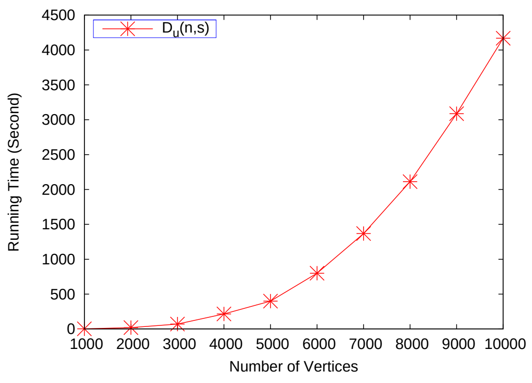


Figure 6.9: Running time of $D_u(n, s)$, for n varies in $\{1000, 2000, \ldots, 10000\}$.

100 in the experiments) where $d(n, s)$ is the number of distinct graphic sequences of length n and sum $s = 2 \times \lceil \frac{n\times(n-1)}{4} \rceil$. The standard deviation for $n = 3$ and $s = 4$ is always 0 because the graphical sequence can only be $(2, 1, 1)$. The standard deviations for lower values of n and s are not stable because of the small numbers of distinct graphic sequences. Empirically, we conservatively estimate the mixing time to be n number of steps.

Figure 6.8 shows the average running time for the generation of 10 graphic sequences of $D_u(n, s)$ for n varying from 100 to 1000 in increments of 100. Figure 6.9 shows the average running time for the generation of 10 graphic sequences of $D_u(n, s)$ for n varying from 1000 to 10000 in increments of 1000. The curves are expected to be of the order of n by construction. Similarly to $D_u(n)$, they reveal the cost of each step in the Markov chain. We can generate a graphic sequence of 10000 elements with sum $49, 995, 000$ uniformly at random in 4200 seconds.

6.6 Summary

In this work, we present four algorithms for the random generation of graphic sequences and prove their correctness. We empirically evaluate their performance.

Two of these algorithms that generate random graphic sequences according to the underlying distribution of random graphs are trivial and as effective and efficient as the corresponding Erdős-Rényi algorithms. We also propose two new algorithms for the uniform random generation of graphic sequences.

To our knowledge these algorithms are the first non trivial algorithms proposed for this task. There is strong evidence that the problem, if not #P-complete, is intrinsically difficult. The algorithms that we propose are Markov chain Monte Carlo algorithms. Our contribution is the original design of the Markov chain and the empirical evaluation of mixing time. Nevertheless the practical problem of generating graphic sequences uniformly at random remains open for large length. We are currently investigating alternative approaches such as local optimization approaches.

Chapter 7

Fast Generation of Random Graphs

7.1 Introduction

Graphs constitute such a versatile data model that they are used in almost every domain of science and humanities. Researchers usually require random graphs with particular properties to evaluate efficiency and effectiveness of algorithms [18], to simulate processes [60] and as the heart of randomized algorithms [87]. In the domain of information and knowledge management, random graphs are used in such applications as web mining and analysis [23, 62], data mining [57] and social network analysis [89] among a multitude of other examples. As a results of the widespread demand for random graphs, random generation of graphs comes to be interesting. Two simple, elegant and general mathematical models are instrumental in random graph generation. The former, noted as $\Gamma_{v,e}$ [33], chooses a graph uniformly at random from the set of graphs with v vertices and e edges. The second model, noted as $\Gamma_{v,p}$ [40], chooses a graph uniformly at random from the set of graphs with v vertices where each edge has the same independent probability p to exist. Nevertheless, both of the two models are referred to as Erdős-Rényi models.

When investigating the former model $\Gamma_{v,e}$, we find that the model is exactly a sampling procedure which samples e edges from a total number of v^2[1] edges. This can be implemented using reservoir algorithm[112]. On the other hand, one can see that $\Gamma_{v,p}$ should behave similar to $\Gamma_{v,e}$ with $e = \binom{v}{2}p$ as v increases. Because in practise $\Gamma_{v,p}$ is the model that is more commonly utilized and easier to analyze, we turn to study the fast generation of $\Gamma_{v,p}$ random graphs.

7.2 The Algorithms

7.2.1 The Baseline Algorithm

We start from the naive algorithm to generate an Erdős-Rényi random graph which is shown as Algorithm 13. We call this algorithm *ER*.

Algorithm 13: ER

```
  Input: $E$ : maximum number of edges; $p$ : inclusion probability
  Output: $G$ : an Erdős-Rényi graph
1 $G = \emptyset$;
2 for $i = 0$ to $E - 1$ do
3     Generate a uniform random number $\theta \in [0, 1)$;
4     if $\theta < p$ then
5         $G \leftarrow e_i$;
6     end
7 end
```

We use a single loop in the algorithm and represent the edges by their indices instead of the standard notion such as (i, j) where i and j are the two vertices of an edge. This makes the algorithm succinct and we can depict the optimization

[1]In the case of a directed graph with self-loops

more easily below. The decoding is orthogonal to the performance of the graph generation algorithm and can be relegated to the stage after graph generation. Thus we omit the presentation of the decoding algorithms. Note that different types of graphs have different decoding algorithms.

7.2.2 ZER

Instead of processing each possible edge in a graph, one can utilize the idea of skipping in the algorithm Z [112] to improve the efficiency. Although this idea has been used by V. Batagelj and U. Brandes [18], we present the complete analysis for consistency and further discussion of optimization. At any step of the edge sequence, let k be the number of edges that are skipped before the next edge is selected. The value of k has a geometric distribution with parameter p. Its probability mass distribution, i.e., the probability that exactly k edges are skipped at any step of the sequence, is:

$$f(k) = (1-p)^k \times p. \tag{7.1}$$

The respective cumulative distribution function, expressing the probability that any number of edges from 0 to k is skipped, is:

$$F(k) = \sum_{i=0}^{k} f(i) = 1 - (1-p)^{k+1}$$

Following the above analysis, we can avoid the per se computation of each skipped edge during the Bernoulli process. Instead, at the beginning of the process, and at any point at which an edge has been selected, we can randomly generate the number of skipped edges k and hence directly select the next $(k+1)^{th}$ edge. In order to generate the value of k, we reason as follows. Let α be a number chosen uniformly at random in $(0, 1]$. Then, the probability that α falls in interval $(F(k-1), F(k)]$ is exactly $F(k) - F(k-1) = f(k)$. In other words, the probability that k is the smallest positive integer such that $\alpha \leq F(k)$ is $f(k)$. Then, in order to assure that each possible value of k is generated with probability $f(k)$, it suffices to generate α and then calculate k as the smallest positive integer such that $F(k) \geq \alpha$, hence $F(k-1) < \alpha \leq F(k)$ (or zero, if no such positive integer exists). Setting $\varphi = 1 - \alpha$, this condition is equivalently written as:

$$1 - (1-p)^k < \alpha \leq 1 - (1-p)^{k+1} \Leftrightarrow \tag{7.2}$$

$$(1-p)^{k+1} \leq \varphi < (1-p)^k \tag{7.3}$$

where φ is chosen uniformly at random in $[0, 1)$. By Equation 7.3, k is computed as

$$k = \max(0, \lceil \log_{1-p} \varphi \rceil - 1)$$

According to the preceding discussion, we give the optimized ER in Algorithm 14. We call this algorithm *ZER*.

Algorithm 14: ZER

```
Input: $E$ : maximum number of edges; $p$ : inclusion probability
Output: $G$ : an Erdős-Rényi graph
1 $G = \emptyset$;
2 $i = -1$;
3 while $i < E$ do
4     Generate a uniform random number $\varphi \in [0, 1)$;
5     Compute the skip value $k = \max(0, \lceil \log_{1-p} \varphi \rceil - 1)$;
6     $i = i + k + 1$;
7     $G \leftarrow e_i$;
8 end
9 Discard the last edge;
```

As argued in [18], a random sampling algorithm that exploits a skipping process is expected to be faster than the algorithm that explicitly considers each candidate sample; accordingly, ZER is expected to be faster than ER, as it considers only a fraction of the total number of edges. However, as we show in Section 7.3, this turns out not to be always the case, due to the logarithm computation overhead (Step 5 of ZER). This motivates us to further improve the efficiency of ZER.

7.2.3 PreZER

The inefficiency of ZER derives from the logarithm computation overhead. We thus try to avoid this overhead based on the following observation. According to Equation 7.1, the probability $f(k)$ that k edges are skipped decreases as a function of k. Figure 7.1 illustrates the $f(k)$ function for several values of p. In effect, the value of k is likely to be lower than some, sufficiently large, fixed integer m. Then, instead of computing the value of k as a function of $\varphi = 1 - \alpha$ at each iteration, we can simply pre-compute the $m + 1$ breakpoints of the intervals in which random number α is most likely to fall, from which the value of k is directly determined.

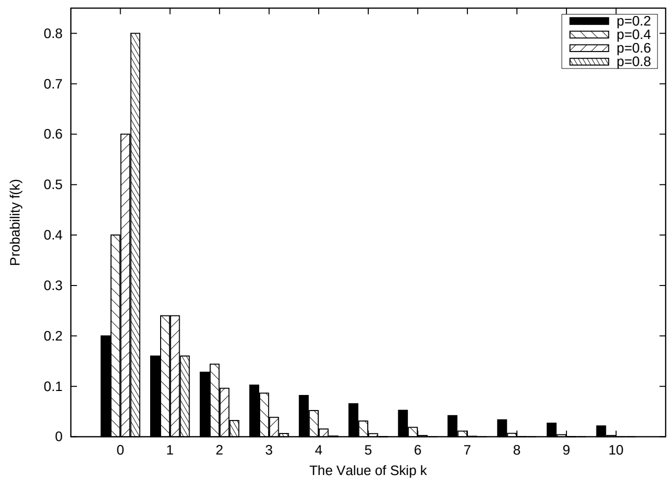


Figure 7.1: f(k) for varying probabilities p.

It then suffices to generate α uniformly at random in the interval $[0, 1)$ and compare it with $F(k)$ for $k = 0$ to m. We can then set the value of k to the smallest value such that $F(k) > \alpha$, or otherwise, if $F(m) \leq \alpha$, compute k by invoking an explicit logarithm computation as in ZER; such computations should be invoked rarely for sufficiently large values of m. The exact value of m is to be decided based on the requirement of the application at hand. Algorithm 15 gives the pseudocode for the complete algorithm. We call this algorithm $PreZER$.

Algorithm 15: PreZER

```
   Input: $E$ : maximum number of edges; $p$ : inclusion probability
   Output: $G$ : an Erdős-Rényi graph
1  $G = \emptyset$;
2  for $i = 0$ to $m$ do
3      Compute the cumulative probability $F[i]$;
4  end
5  $i = -1$;
6  loop:
7  while $i < E$ do
8      Generate a uniform random number $\alpha \in (0, 1]$;
9      $j = 0$;
10     while $j \leq m$ do
11         if $F[j] > \alpha$ then
12             Set the skip value $k = j$; Break;
13         $j = j + 1$;
14     end
15     if $j = m + 1$ then
16         Compute the skip value $k = \lceil \log_{1-p}(1-\alpha) \rceil - 1$;
17     end
18     $i = i + k + 1$;
19     $G \leftarrow e_i$;
20 end
21 Discard the last edge;
```

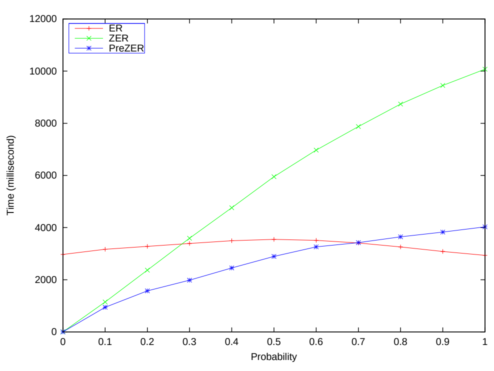


Figure 7.2: Running times for all algorithms.

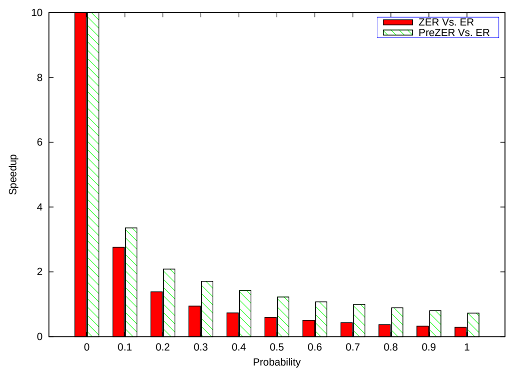


Figure 7.3: Speedups for ZER and PreZER over ER.

7.3 Performance Evaluation

We compare the running times of ER, ZER and PreZER based on various edge inclusion probabilities and graph sizes. We also show the speedup that ZER and PreZER gain against ER. All the experiments are run on a 2.33GHz Core 2 Duo CPU machine with 4GB of main memory under Windows XP. All the algorithms are implemented using Visual C++ 9.0.

7.3.1 Varying probability

We run the three algorithms to generate directed random graphs with self-loops. The number of vertices is set to 10,000 and the edge inclusion probability varies from 0 to 1, with step 0.1. The m parameter for PreZER is set to 9, a value which is empirically determined to yield the best performance. We runs each algorithm 10 times and take the average running time. We observe from Figure 7.2 that both ZER and PreZER become faster as the value of p decreases. This is expected, given that smaller values of p offer more opportunities to skip edges. Still, these two relatively naive edge-skipping algorithms are not consistently faster than ER. ZER remains faster than ER only for probability values up to p = 0.3; this poor

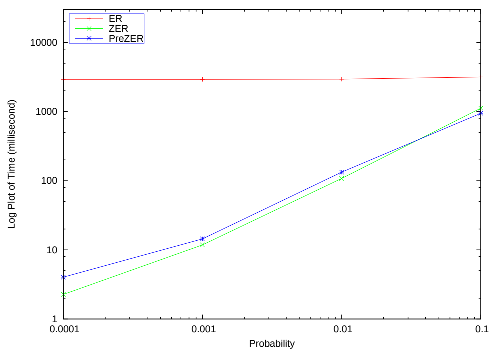


Figure 7.4: Running times for small probabilities.

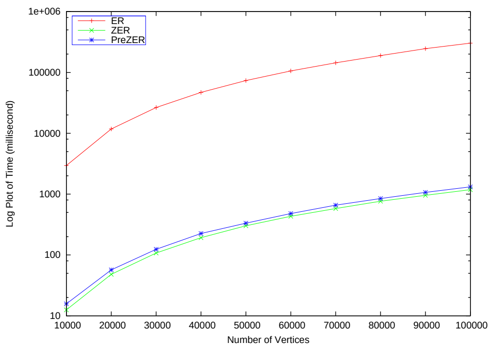


Figure 7.5: Runtime for varying graph size, p = 0.001.

performance is explained by the overhead due to logarithm computations. PreZER is more efficient than ER for probability values up to p = 0.7; after that point, the advantage of having pre-computed values does not suffice to achieve better performance. Figure 7.3 gives the speedups gained by ZER and PreZER over ER. The average speedup for ZER and PreZER are 1 and 1.5, respectively. Besides, for $p \leq 0.5$, the average speedup for ZER and PreZER are 1.3 and 2 respectively. We can also observer that the speedup for small p turns to be more significant.

In practice, the value p for $\Gamma_{v,p}$ is usually very small, say less than 0.1. We thus evaluate the running times for small values of p. We set p varying in $\{0.0001, 0.001, 0.01, 1\}$ and plot Figure 7.4. Because of the enormous number of skips, ZER and PreZER highly improve the efficiency of ER for small probabilities. The difference between ZER and PreZER becomes subtle for smaller values of p because skipping dominates the graph generation.

7.3.2 Varying graph size

We also address the question of scalability of our algorithms to larger graph sizes. We vary the value of p in $\{0.001, 0.01, 0.1\}$ and for each p we vary the graph size from 10,000 to 100,000, with step 10,000. Figure 7.5, 7.6 and 7.7 show the

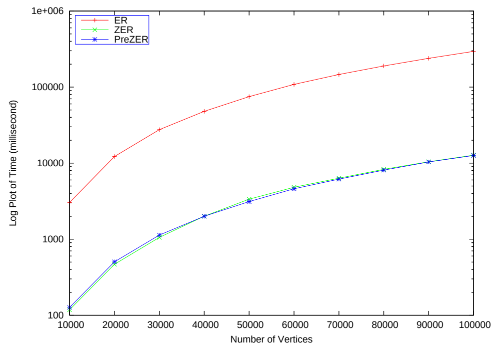


Figure 7.6: Runtime for varying graph size, p = 0.01.

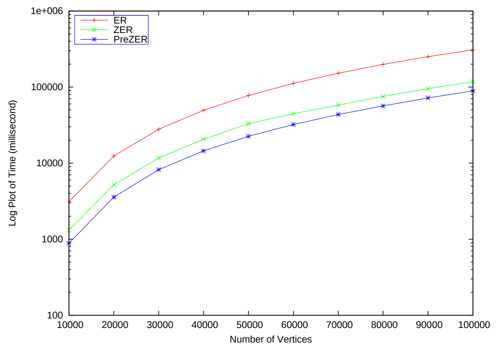


Figure 7.7: Runtime for varying graph size, p = 0.1.

execution time results. They verify that the difference between ZER and PreZER is attenuated for smaller values of p.

7.4 Summary

This work has led to the proposal of a new algorithm for the generation of random graphs in the Erdős-Rényi model $\Gamma_{v,p}$, namely PreZER. In order to motivate, explain, and evaluate this work, we have outlined a succession of algorithms leading to our contributions. The baseline algorithm, ER, and the two enhancements thereupon, namely ZER and PreZER,are all sequential algorithms. To our knowledge, PreZER is the fastest known sequential algorithm. Further improvement can be achieved by implementing parallel algorithms on multi-core processors such as GPU. However, this is beyond the scope of this thesis. We discuss this issue and present the corresponding experiment results in Appendix B. The details can be found in [90].

Chapter 8

Future Work

We will first complete the work in sampling random induced subgraphs. We have empirically show that the neighbour reservoir sampling algorithm can generate nearly uniform connected induced subgraphs of a given size. On the other hand, we will theoretically analyze the probability distribution of the subgraphs produced by this algorithm. The analysis will make this work more steady and, probably give us an opportunity to find an efficient algorithm that produces a true uniform distribution.

In the domain of sampling streams of continuous data, we will consider bias sampling. In many applications, frequent data are more interesting than infrequent ones. Therefore, for a given data stream with unknown length, we consider the problem of maintaining a fixed-size random sample, such that each data in the sample is selected with probability proportional to its number of occurrence in the entire data stream when it is processed. This sampling scheme will generally sample more frequent data. Actually, by adjusting the proportion of sampling probability we can generate a sample having bias either to frequent data or infrequent data, depending on different applications.

In the domain of graph sampling, we will consider sampling dynamic graphs by taking into account vertex deletion and edge deletion. As we discussed in this thesis, existing graph sampling algorithms are designed to sample from static graphs. However, the real life graphs encountered in modern applications are dynamic. To avoid re-sampling from scratch, we have proposed incremental algorithms for sampling dynamic graph with vertex addition and edge addition. The next step is to develop algorithms for the scenario of vertex deletion and edge deletion.

In the domain of graph generation, we find that although a lot of models have been proposed, none of them is dedicated for bipartite graph generation. In a bipartite graph, the vertices are divided into two sets. Edges can only link two vertices that belong to different sets. In our recent study, we find that some real bipartite graphs exhibit power-law degree distribution for both sets of vertices, respectively. Also, the evolution of density and average shortest path length show some interesting results. We discuss one such example in Appendix D. We will study more real bipartite graphs and find common properties among them. Then we are interested in devising models for generating random bipartite graphs.

Chapter 9

Conclusion

In this thesis, we discuss random sampling and generation problems over data streams and graphs. We systematically review existing problems and the state-of-the-art algorithms. We then propose novel algorithms to solve five problems. We revisit the main results as follows.

- We propose the FIFO sampling algorithm to sample a data stream with a sliding window. FIFO always maintains a nearly uniform random sample of size n from the w most recent data, where $1 \leq n \leq w$. FIFO overcomes the drawback of periodic sample design of the simple algorithm, and the drawback of unbounded memory usage of the chain-sample algorithm. However, FIFO may sample a few expired data. This is because we did not deterministically exclude expired data in our algorithm. Nevertheless, experiment results show FIFO is both effective and efficient for sampling a data stream with a sliding window.

- We propose the NRS algorithm to sample representative subgraphs from original large graphs. NRS samples connected induced subgraphs of size k uniformly at random from an original graph of size n, where $1 \leq k \leq n$. To the best of our knowledge, we are the first to propose algorithms for sampling connected induced subgraphs. Experimental results show NRS successfully realizes the compromise between effectiveness and efficiency, compared with ARS and MHS. Also, the samples generated by NRS capture important metric of the original graphs better than state-of-art algorithms. This result suggests that connected induced subgraphs inherently preserve the properties of their original graphs and may lead to a new direction of graph sampling. However, the analytical distribution of samples generated by NRS is unknown and still needs to be worked out.

- We propose the IMS and SMS algorithms to incrementally sample from dynamic graphs. We consider vertex addition and edge addition in a dynamic graph. The algorithms incrementally maintain a representative sample graph of size n of a dynamic original graph. Both of the algorithms incrementally apply the idea of Metropolis Graph Sampling. Experiment results show that our proposed algorithms realize a compromise between effectiveness and efficiency of the baseline algorithms. The algorithms is practical for the incremental sampling of the large dynamic graphs being created on the Internet, the World Wide Web and its social media. The next step is to devise algorithms that cater for vertex deletion and edge deletion in the original graph.

- We proposed MCMC algorithms to generate random graphic sequences. We constructed the Markov chain, analyzed its stationary distribution and empirically evaluated the mixing time. To the best of our knowledge, we are the first to study the problem of generating random graphic sequences. Experimental results show our algorithms effectively solve the problem. Our algorithms may be used as fundamental tools for random graph generation with prescribed graphic sequences. However, generating long random graphic sequences with our algorithm is time-consuming. This is due to the high time complexity of realizability testing of graphic sequences.

- We propose the PreZER algorithm for fast generation of Erdős-Rényi random graphs. The algorithm is based on a pre-computation of skip. Experiment results show significant speedup for PreZER with respect to the baseline algorithm. To our knowledge, PreZER is the fastest known sequential algorithm. Further improvements can be achieved by paralleling the proposed algorithms, for instance, on GPU.

We conceptually point out the inherent connection between random sampling and generation, and introduce the conception of construction, enumeration and counting. To some extent, our work is motivated by the malpractice of construction, enumeration and counting in finding representative samples of large-scale datasets. However, we are interested in revealing the intrinsic relations between the former and the latter problems, i.e., the transition from construction, enumeration and counting, to random sampling and generation.

Bibliography

[1] http://blog.twitter.com/2011/03/numbers.html.

[2] CUDA Zone: Toolkit & SDK. http://www.nvidia.com/object/what_is_cuda_new.html.

[3] Graph API. http://developers.facebook.com/docs/reference/api.

[4] Old REST API. http://developers.facebook.com/docs/reference/rest.

[5] RestFB. http://restfb.com.

[6] Stanford Network Analysis Project. http://snap.stanford.edu/index.html.

[7] Twitter. http://blog.twitter.com/2011/06/200-million-tweets-per-day.html.

[8] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, pages 607–618, 2006.

[9] N. K. Ahmed, F. Berchmans, J. Neville, and R. Kompella. Time-based sampling of social network activity graphs. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 1–9. ACM, 2010.

[10] Y.-Y. Ahn, S. Han, H. Kwak, S. B. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *WWW*, pages 835–844, 2007.

[11] E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973.

[12] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *SIGMOD Conference*, pages 685–696, 2011.

[13] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD Conference*, pages 539–550, 2003.

[14] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.

[15] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, Oct. 1999.

[16] T. M. Barnes and C. D. Savage. A recurrence for counting graphical partitions. *Electr. J. Comb.*, 2, 1995.

[17] T. M. Barnes and C. D. Savage. Efficient generation of graphical partitions. *Discrete Applied Mathematics*, 78(1-3):17–26, 1997.

[18] V. Batagelj and U. Brandes. Efficient generation of large random networks. *Physcal Review E*, 71, 2005.

[19] I. Bezáková, D. Stefankovič, V. V. Vazirani, and E. Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. In *In Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA*, pages 900–907, 2006.

[20] P. Biernacki and D. Waldorf. Snowball Sampling: Problems and Techniques of Chain Referral Sampling. *Sociological Methods and Research*, 10(2), 1981.

[21] N. Biggs. *Algebraic Graph Theory.* Cambridge University Press, 1993.

[22] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *PODS*, pages 147–156, 2009.

[23] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[24] J. M. Burns. *The number of degree sequences of graphs.* PhD thesis, Massachusetts Institute of Technology, June 2007.

[25] S. Chaudhuri, G. Das, and V. R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2):9, 2007.

[26] F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Ann. Comb.*, 6(2):125–145, Nov. 2002.

[27] A. Clauset, C. Moore, and M. E. J. Newman. Structural inference of hierarchies in networks. *CoRR*, abs/physics/0610051, 2006.

[28] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, pages 449–460, 2004.

[29] C. Cooper, A. M. Frieze, and M. Krivelevich. Hamilton cycles in random graphs with a fixed degree sequence. *SIAM J. Discrete Math.*, 24(2):558–569, 2010.

[30] T. M. Cover and J. A. Thomas. *Elements of information theory.* Wiley-Interscience, New York, NY, USA, 1991.

[31] M. K. Cowles and B. P. Carlin. Markov chain monte carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91:883–904, 1996.

[32] P. Erdős and T. Gallai. Graphs with prescribed degrees of vertices (Hungarian). *Mat. Lapok*, 11:264–274, 1960.

[33] P. Erdős and A. Rényi. On random graphs. I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[34] K. Fukuda and T. Matsui. Finding all the perfect matchings in bipartite graphs. *Appl. Math. Lett*, 7:15–18, 1989.

[35] A. Gelman and D. B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457–472, 1992.

[36] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD Conference*, pages 379–392, 2008.

[37] R. Gemulla, W. Lehner, and P. J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *VLDB*, pages 595–606, 2006.

[38] C. I. D. Genio, H. Kim, Z. Toroczkai, and K. E. Bassler. Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *CoRR*, abs/1002.2975, 2010.

[39] J. Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *IN BAYESIAN STATISTICS*, pages 169–193. University Press, 1992.

[40] E. N. Gilbert. Random graphs. *Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[41] W. Gilks and D. Spiegelhalter. *Markov chain Monte Carlo in practice.* Chapman & Hall/CRC, 1996.

[42] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. In *KDD*, pages 167–176, 2006.

[43] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, pages 2498–2506, 2010.

[44] C. Gkantsidis, M. Mihail, and E. W. Zegura. The markov chain simulation method for generating connected power law random graphs. In *ALENEX*, pages 16–25, 2003.

[45] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In *SIGMOD Conference*, pages 243–252, 1994.

[46] C. Hahn, S. Warren, U. S. D. of Energy. Environmental Sciences Division, and C. D. I. A. C. (U.S.). *Extended edited synoptic cloud reports from ships and land stations over the globe, 1952-1996.* Environmental Sciences Division publication. Printed for the Environmental Sciences Division, Office of Biological and Environmental Research, United States Dept. of Energy by the Carbon Dioxide Information Analysis Center, Oak Ridge National Laboratory, 1999.

[47] S. L. Hakimi. *J. Soc. Indust. Appl. Math.*, pages 496–506.

[48] M. Hansen and W. Hurwitz. On the theory of sampling from finite populations. *Annals of Mathematical Statistics*, 14, 1943.

[49] F. Harary and E. M. Palmer. *Graphical Enumeration*, volume 16. 1973.

[50] M. A. Hasan and M. J. Zaki. Musk: Uniform sampling of k maximal patterns. In *SDM*, pages 650–661, 2009.

[51] M. A. Hasan and M. J. Zaki. Output space sampling for graph patterns. *PVLDB*, 2(1):730–741, 2009.

[52] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, Apr. 1970.

[53] V. Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat.*, 80:477–480, 1955.

[54] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. *Computer Networks*, 33(1-6):295–308, 2000.

[55] D. Horn. Stream reduction operations for GPGPU applications. In *GPU Gems 2*. Addison Wesley, 2005.

[56] C. Hübler, H.-P. Kriegel, K. M. Borgwardt, and Z. Ghahramani. Metropolis algorithms for representative subgraph sampling. In *ICDM*, pages 283–292, 2008.

[57] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[58] K. E. Iverson. *A programming language.* Wiley, 1962.

[59] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *KDD*, pages 367–370, 1996.

[60] M. Kaiser, R. Martin, P. Andras, and M. P. Young. Simulation of robustness against lesions of cortical networks. *European Journal of Neuroscience, 25:3185–3192, 2007*, (arXiv:0704.0392), Aug 2008.

[61] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[62] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web as a graph: Measurements, models, and methods. In *COCOON*, pages 1–17, 1999.

[63] K. Klemm and V. M. Eguiluz. Growing Scale-Free Networks with Small World Behavior, 2001.

[64] K. Klemm and V. M. Eguíluz. Highly Clustered Scale-free Networks. *Physical Review E*, 65, 2002.

[65] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., 1997.

[66] H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[67] W. B. Langdon. A fast high-quality pseudo-random number generator for graphics processing units. In *IEEE Congress on Evolutionary Computation*, pages 459–465, 2008.

[68] W. B. Langdon. A fast high-quality pseudo-random number generator for nVidia CUDA. In *GECCO (Companion)*, pages 2511–2514, 2009.

[69] A. Leon-Garcia. *Probability, Statistics, and Random Processes for Electrical Engineering.* Prentice Hall, 2008.

[70] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD*, pages 631–636, 2006.

[71] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[72] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times.* American Mathematical Society, 2006.

[73] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, pages 93–106, 2008.

[74] X. Lu and S. Bressan. Generating random graphic sequences. In *DASFAA*, pages 570–579, 2011.

[75] X. Lu and S. Bressan. Sampling connected induced subgraphs uniformly at random. In *SSDBM*, pages 195–212, 2012.

[76] X. Lu and S. Bressan. Incremental algorithms for sampling dynamic graphs. In *DEXA*, 2013.

[77] X. Lu, G. Cheliotis, X. Cao, Y. Song, and S. Bressan. The configuration of networked publics on the web: evidence from the greek indignados movement. In *WebSci*, pages 185–194, 2012.

[78] X. Lu, Y. Song, and S. Bressan. Fast identity anonymization on graphs. In *DEXA*, pages 281–295, 2012.

[79] X. Lu, W. H. Tok, C. Raïssi, and S. Bressan. A simple, yet effective and efficient, sliding window sampling algorithm. In *DASFAA*, pages 337–351, 2010.

[80] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *WWW*, pages 701–710, 2010.

[81] S. Maslov and K. Sneppen. Specificity and Stability in Topology of Protein Networks. *Science*, 296(5569):910–913, 2002.

[82] A. I. McLeod and D. R. Bellhouse. Miscellanea: A convenient algorithm for drawing a simple random sample. 32(2):182–184, 1983.

[83] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1953.

[84] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. May 2004.

[85] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, Oct. 2002.

[86] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6:161–179, 1995.

[87] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, New York, NY, USA, 1995.

[88] M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701, 2002.

[89] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proc. Natl. Acad. Sci. USA*, 99:2566–2572, 2002.

[90] S. Nobari, X. Lu, P. Karras, and S. Bressan. Fast random graph generation. In *EDBT*, pages 331–342, 2011.

[91] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[92] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

[93] U. N. Peled and M. L. Srinivasan. The polytope of degree sequences. *Linear Algebra Appl.*, 114:349–377, 1989.

[94] N. Przulj, D. G. Corneil, and I. Jurisica. Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics*, 22(8):974–980, 2006.

[95] C. Raïssi and P. Poncelet. Sampling for sequential pattern mining: From static databases to data streams. In *ICDM*, pages 631–636, 2007.

[96] S. Ross. *First Course in Probability, A (8th Edition).* Prentice Hall, 8 edition, Jan. 2009.

[97] F. Ruskey, R. Cohen, P. Eades, and A. Scott. *Congressus Numerantium.*

[98] G. Sierksma and H. Hoogeveen. Seven criteria for integer sequences being graphic. *Journal of Graph Theory*, 15(2):223–231, 1991.

[99] A. Silberstein, R. Braynard, C. S. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, page 68, 2006.

[100] A. Sinclair. *Algorithms for random generation and counting: a Markov chain approach.* Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.

[101] S. Skiena. *Implementing discrete mathematics - combinatorics and graph theory with Mathematica.* Addison-Wesley, 1990.

[102] N. J. A. Sloane. The encyclopedia of integer sequences, 1995.

[103] R. Stanley. A zonotope associated with graphical degree sequences. In *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift.* ACM, AMS, 1991.

[104] B. R. Tamma, B. S. Manoj, and R. R. Rao. Time-based sampling strategies for multi-channel wireless traffic characterization in tactical cognitive networks. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, Nov. 2008.

[105] Y. Theodoridis, J. Silva, and M. Nascimento. On the Generation of Spatiotemporal Datasets. In *Advances in Spatial Databases*, volume 1651, pages 147–164. 1999.

[106] H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145, 1996.

[107] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[108] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[109] V. V. Vazirani. *Approximation Algorithms.* Berlin: Springer, 2003.

[110] A. Vázquez, J. Oliveira, and A. Barabási. Inhomogeneous evolution of subgraphs and cycles in complex networks. *Physical Review E*, 2005.

[111] F. Viger and M. Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. In *COCOON*, pages 440–449, 2005.

[112] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[113] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[114] S. Wu, J. M. Hofman, W. A. Mason, and D. J. Watts. Who says what to whom on twitter. WWW '11, pages 705–714, 2011.

[115] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of Sampling for Data Mining of Association Rules. Technical Report TR 617, University of Rochester, Computer Science Department, 1996.

Appendix A

Sharp-P-Complete Problems

Valiant [108] discussed the problem of computing the permanent of a given $(0, 1)$-matrix. The permanent of an $n \times n$ matrix $A = (a_{i,j})$ is defined as

$$perm(A) = \sum_{\sigma} \prod_{i=1}^{n} a_{i,\sigma(i)},$$

where the summation is over the $n!$ permutations of $(1, 2, \ldots, n)$. They proved that the problem is $\sharp P$-complete. This paper also introduced $\sharp P$ as a complexity class for the first time.

An equivalent problem of the permanent problem is the problem of computing the number of perfect matchings in a bipartite graph [34, 91]. A matching of a graph is a set of non-adjacent edges, that is, no two edges share a common vertex. A perfect matching of a graph is a matching that contains all the vertices of the graph. The problem is also $\sharp P$-complete.

Another famous problem that is proved to be $\sharp P$-complete is the *chromatic polynomial* problem [21]. The problem counts the number of graph colorings using no more than k colors for a particular graph G. The problem of graph coloring is to color the vertices (edges) of a graph using k colors, such that no two adjacent vertices (edges) share the same color.

Other examples of $\sharp P$-complete problem include computing the number of variable assignments satisfying a given $\sharp SAT$ formula, computing the number of variable assignments satisfying a given DNF formula, computing the number of different topological orderings for a given directed acyclic graph, etc.

If there exists a polynomial-time algorithm that solves any $\sharp P$-complete problem, it would imply that $P = NP$. Till now no such algorithm is found. However, many $\sharp P$-complete problem have a *fully polynomial-time randomized approximation scheme* [109], or "FPRAS" for short. The scheme can produce approximation results with high probability to an arbitrary degree of accuracy, in polynomial time w.r.t. both the size of the problem and the degree of accuracy required. For example, Bezáková et al. [19] propose an accelerating simulated annealing algorithm to count the number of perfect matchings in a bipartite graph. The algorithm belongs to FPRAS.

Appendix B

Parallel Graph Generation Using GPU

We leverage the parallel-processing capabilities of a Graphics Processing Unit to develop three successive data parallel algorithms for random graph generation in the $\Gamma_{v,p}$ model. These algorithms are the data parallel counterparts of ER, ZER and PreZER in Section 7.

We use nVidia graphic cards as our implementation platform. In order to program the GPU, we use the C-language Compute Unified Device Architecture (CUDA) [2] parallel-computing application programming interface. CUDA is provided by nVidia and works on nVidia graphic cards.

We use Langdon's pseudo-random number generator [67, 68], a data-parallel version of Park-Miller's pseudo-random number generator [92], to generate random numbers on GPU. We implement the *Prefix Sum* algorithm [58] to create a sequence of partial sums from an existing sequence of numbers. These partial sums are used to determine the location of selected edges. We also employ stream compaction, a method that compresses an input array A into a smaller array B by keeping only

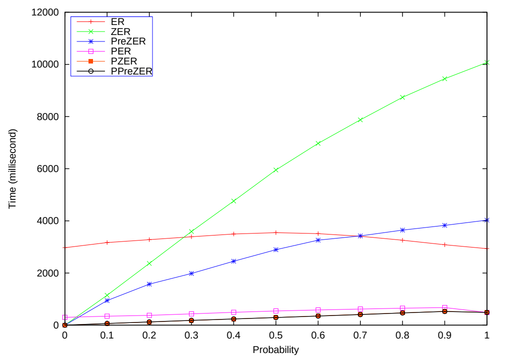


Figure B.1: Running times for all the algorithms.

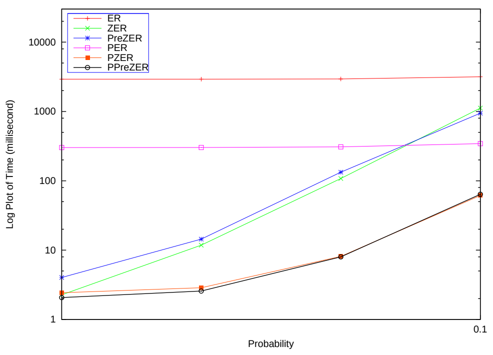


Figure B.2: Running times for small probabilities.

the elements that verify a predicate p, while preserving the original relative order of the elements [55].

With above architectures and primitives, we implement three parallel counterpart algorithms of ER, ZER and PreZER. We call them PER, PZER and PPreZER, respectively. We then evaluate the performance of all the six algorithms we have presented and introduced to each other. The parallel algorithms run on the same machine on which we run our sequential algorithms, with a GeForce 9800 GT graphics card having 1024MB of global memory, 14 streaming processors and a PCI Express $\times 16$ bus.

Still, we set the algorithms to generate directed random graphs with self loops, having 10,000 vertices, hence at most 100,000,000 edges. We measure execution time as user time, averaging the results over ten runs.

We first turn our attention to the execution time of the six algorithms as a function of inclusion probability p. Figure B.1 shows the results, with the best parameter settings in those algorithms where they are applicable.

All three parallel algorithms are significantly faster than the sequential ones for all values of p; the only exception to this observation is that PER is slightly slower than ZER and PreZER for very small values of p, as it has to generate E random

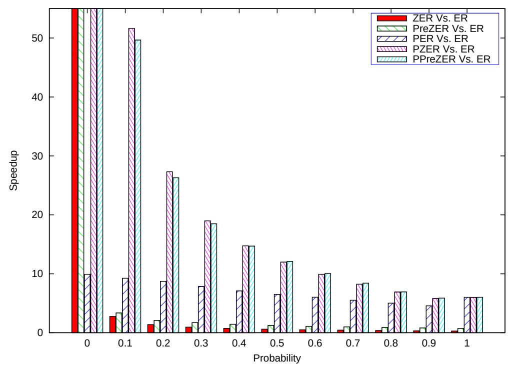


Figure B.3: Speedup for all algorithms over ER.

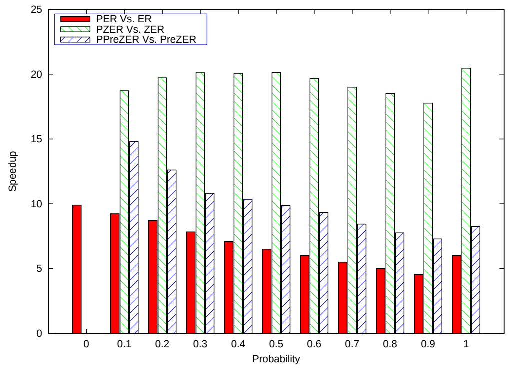


Figure B.4: Speedup for parallel algorithms over their sequential counterparts.

numbers whatever the value of p. Figure B.2 illustrates the effect of such small values of p on execution time on logarithmic axes.

Next, Figure B.3 presents the average speedup over the baseline ER algorithm, for the other five algorithms in the comparison, as a function of p. The average speedup for ZER, PreZER, PER, PZER and PPreZER are 1, 1.5, 7.2, 19.3, 19.2, respectively. Besides, for $p \leq 0.5$, the average speedup for ZER, PreZER, PER, PZER and PPreZER are 1.3, 2, 8.4, 29.9 and 29.4, respectively.

In addition, in Figure B.4 we gather the average speedup of each parallel algorithm over its sequential counterparts. The average speedup for PER, PZER and PPreZER over their sequential version are 7.2, 22.8 and 11.7, respectively. For $p \leq 0.5$ the average speedup for PER, PZER and PPreZER are 8.4, 23.4 and 13.7, respectively.

We then further compare the three parallel algorithms in our study. Figure B.5 shows the overall execution times for these three algorithms only. For all probability values, PZER and PPreZER are faster than PER. PPreZER is slightly faster than PZER for probabilities greater than 0.4 and slightly slower or identical for the rest. This result arises from the handling of branching conditions by the GPU, as

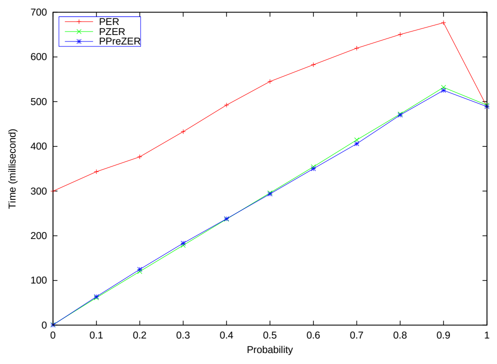


Figure B.5: Running times for parallel algorithms.

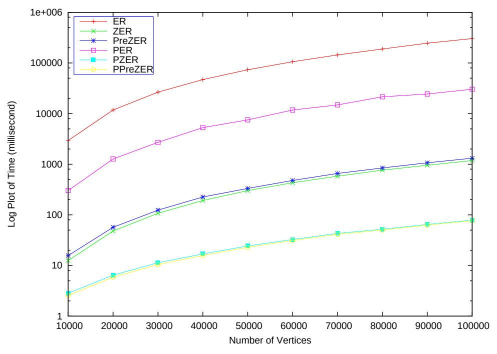


Figure B.6: Runtime for varying graph size, $p = 0.001$.

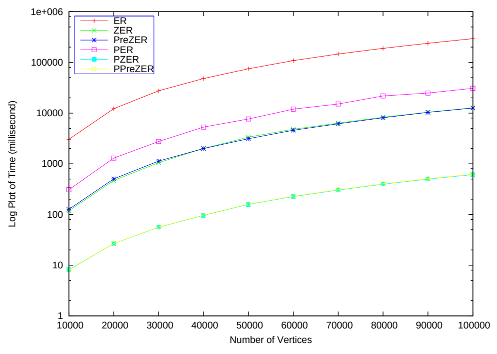


Figure B.7: Runtime for varying graph size, $p = 0.01$.

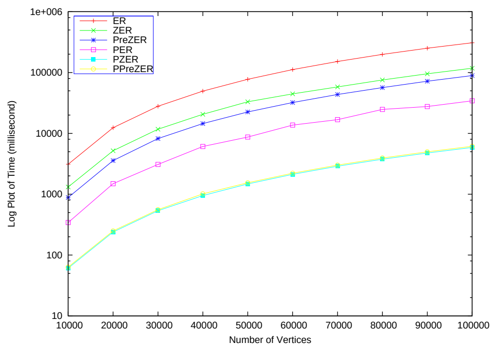


Figure B.8: Runtime for varying graph size, $p = 0.1$.

concurrent threads taking different execution paths are serialized.

We also address the question of scalability of our algorithms to larger graph sizes. Figures B.6, B.7 and B.8 show the execution time results for $p = 0.001$, $p = 0.01$ and $p = 0.1$, as a function of increasing number of vertices.

The results reconfirm our previous findings and carry them forward to larger graph structures in terms of vertices. They verify that the difference between ZER and PreZER is attenuated for smaller values of p, while the advantages of skip-based parallel algorithm are amplified for such smaller probability values.

In summary, the three algorithms PER, PZER and PPreZER are data parallel versions of their sequential counterparts designed for graphics cards and implemented in CUDA. To our knowledge, PreZER is the fastest known sequential algorithm, while PZER and PPreZER can both claim the title of the fastest known parallel algorithms for a GPU. They yield average speedups of 1.5 and 19 over the baseline algorithm, respectively.

Appendix C

Fast Identity Anonymization on Graphs

In this work, we aim to improve the algorithms proposed in [73] by Liu and Terzi. They propose the notion of k-degree anonymity to address the problem of identity anonymization in graphs. A graph is k-degree anonymous if and only if each of its vertices has the same degree as that of, at least, k-1 other vertices. The anonymization problem is to transform a non-k-degree anonymous graph into a k-degree anonymous graph by adding or deleting a minimum number of edges.

Liu and Terzi propose an algorithm that remains a reference for k-degree anonymization. The algorithm consists of two phases. The first phase anonymizes the degree sequence of the original graph. The second phase constructs a k-degree anonymous graph with the anonymized degree sequence by adding edges to the original graph. We call this algorithm the *K-degree Anonymization (KDA)*. The algorithm finds a theoretically optimal solution for anonymizing a graph. However, we observe that the algorithm is not efficient and not effective for real large graphs. The reasons are as follows. First, the dynamic programming algorithm in the degree-anonymization phase cannot construct a realizable degree sequence in a small number of iterations. As the testing of realizability of the degree sequence in each iteration is very time-consuming, the efficiency of the entire algorithm is highly affected. Second, the graph-construction phase on real large graphs invokes the `Probing` function too many times. As the function adds noise to the original degree sequences, the effectiveness of the entire algorithm is affected. Also, the testing of realizability is conducted every time after `Probing` is invoked. Therefore the efficiency is reduced once again.

Motivated by the above observations, we study fast k-degree anonymization on graphs at the risk of marginally increasing the cost of degree anonymization, i.e., the edit distance between the anonymized graph and the original graph.

We propose a greedy algorithm that anonymizes the original graph by simultaneously adding edges to the original graph and anonymizing its degree sequence. We thereby avoid realizability testing by effectively interleaving the anonymization of the degree sequence with the construction of the anonymized graph in groups of vertices. The algorithm consists of three steps in each iteration, namely, `greedy_examination`, `edge_creation`, and `relaxed_edge_creation`.

The `greedy_examination` step determines the number of consecutive vertices that are going to be anonymized. The `edge_creation` step anonymizes the vertices found by `greedy_examination`. The `edge_creation` step relaxes the anonymizing condition if `edge_creation` cannot find a valid solution, and always outputs a k-anonymized degree sequence. We call this algorithm the *Fast K-degree Anonymization (FKDA)* algorithm. The details of the algorithm can be found in [78].

We implement KDA and three variants of FKDA, FKDA 1, FKDA 2 and FKDA 3, corresponding to the three heuristics in C++. We run all the experiments on a cluster of 54 nodes, each of which has a 2.4GHz 16-core CPU and 24 GB memory.

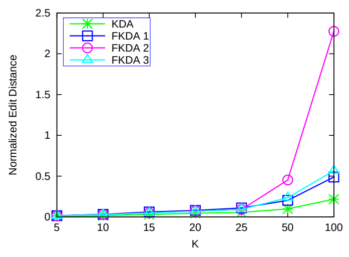


Figure C.1: ED: Email-Urv.

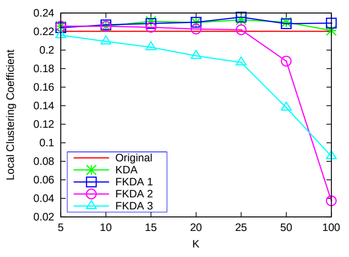


Figure C.2: CC: Email-Urv.

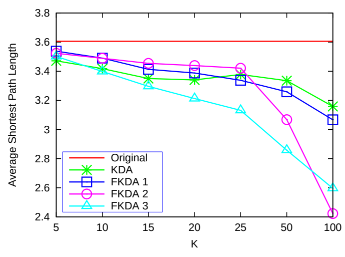


Figure C.3: ASPL: Email-Urv.

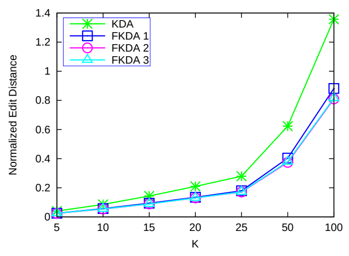


Figure C.4: ED: Wiki-Vote.

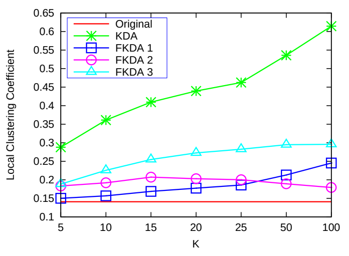


Figure C.5: CC: Wiki-Vote.

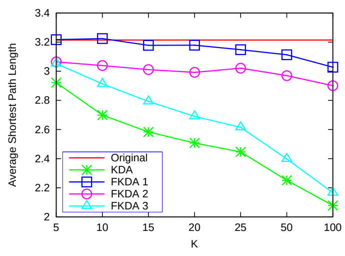


Figure C.6: ASPL: Wiki-Vote.

We use three datasets, namely, **Email-Urv**, **Wiki-Vote** and **Email-Enron**. We conducts experiments on these three graphs. The different sizes of the three graphs illustrate the performance of KDA and FKDA on small (1133 vertices), medium (7115 vertices) and relatively large (36692 vertices) graphs.

We compare the effectiveness of the algorithms by evaluating the variation of several utility metrics: edit distance (ED), clustering coefficient (CC) and average shortest path length (ASPL) (following [73]).

We vary the value of k in the range $\{5, 10, 15, 20, 25, 50, 100\}$. For each value of k, we run each algorithm 10 times on each dataset and compute the average value of the metrics.

Figure C.1-C.3, C.4-C.6 and C.7-C.9 show the results on Email-Urv, Wiki-Vote and Email-Enron, respectively.

We see that FKDA produces less similar results with that in the original graphs on Email-Urv and more similar results on Wiki-Vote and Email-Enron than KDA

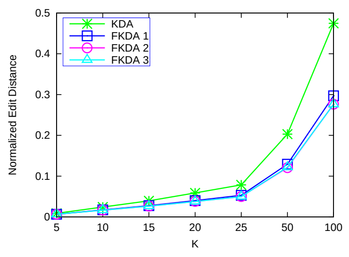


Figure C.7: ED: Email-Enron.

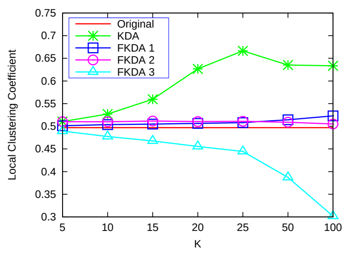


Figure C.8: CC: Email-Enron.

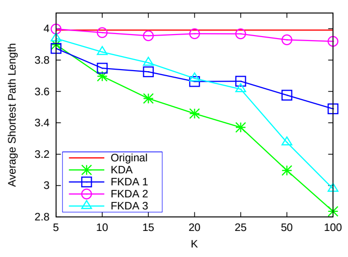


Figure C.9: ASPL: Email-Enron.

does. This is because on small graphs KDA can construct a realizable degree sequence with a small number of repetition of `Probing`, whereas on large graphs KDA invokes `Probing` a large number of times before a realizable degree sequence is constructed. Moreover, `Probing` randomly adds noise to the original degree sequence, while `relaxed_edge_creation` increases a small degree only if the corresponding vertex can be wired to an anonymized vertex with residual degree. The noise added to the original degree sequence is minimized. Thus `Probing` adds more noise than `relaxed_edge_creation` does to the degree sequences of the large graphs. Overall, FKDA adds less edges than KDA does to the two larger graphs.

We further compare the performances of the three variants of FKDA. The overall results show that FKDA 1 and FKDA 2 preserve the utilities of the original graph better than FKDA 3 does. Nevertheless, FKDA 3 has an interesting property that it can generate a random k-degree anonymous graph.

We compare the efficiency of the algorithms by measuring their execution time.

We vary the value of k in the range $\{5, 10, 15, 20, 25, 50, 100\}$. For each value of k, we run each algorithm 10 times on each dataset and compute the average execution time. We also compute the speedup of FKDA versus KDA for each parameter setting.

We see that FKDA is significantly more efficient than KDA. The speedup varies from hundreds to one million on different graphs. The inefficiency of KDA is due to

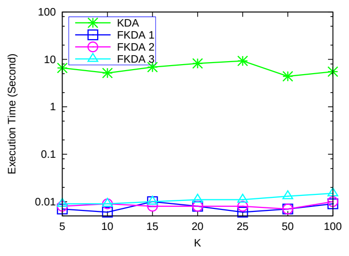


Figure C.10: Execution time on Email-Urv.

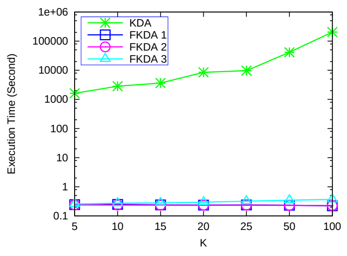


Figure C.11: Execution time on Wiki-Vote.

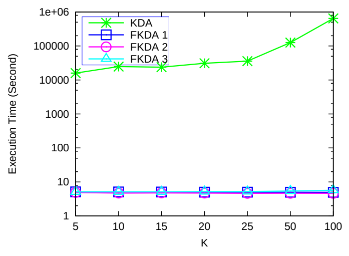


Figure C.12: Execution time on Email-Enron.

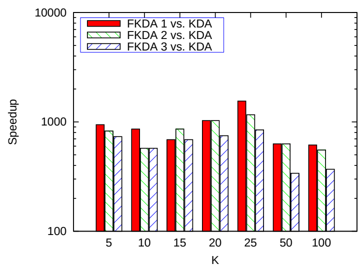


Figure C.13: Speedup of FKDA vs. KDA on Email-Urv.

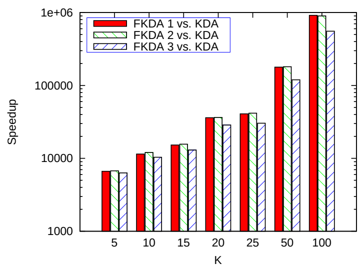


Figure C.14: Speedup of FKDA vs. KDA on Wiki-Vote.

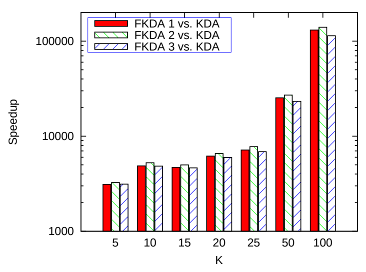


Figure C.15: Speedup of FKDA vs. KDA on Email-Enron.

the decoupling of the checking of realizability of the anonymized degree sequences from the construction of graph.

In summary, our algorithm results in larger edit distance on small graphs but smaller edit distance on large graphs compared with the algorithm of Liu and Terzi. Our algorithm is much more efficient than the algorithm of Liu and Terzi.

Appendix D

Bipartite Graphs of the Greek Indignados Movement on Facebook

This work studies the use of online media in social movement. The details can be found in [77]. We focus our attention on the anti-austerity movement of the Greek 'Indignados', also known as the 'aganaktismeni', or 'αγανακτισμένοι' in Greek. As Facebook is often reported in the media as a central component of the communications strategy of the Greek indignados and other movements with similar characteristics, we focus on identifying Facebook pages that are related to the events that unfolded in Greece, by utilizing a set of keywords.

We use the *RestFB* package [5] to collect the data. This package in turn consists of the *Facebook Graph API* [3] and the *Old REST API* [4] client written in Java. The Facebook Graph API gives access to historical data for a period of our choice.

We compile a list of the pages, groups and events that are returned by the search function of the Facebook Graph API when given each of the following keywords:

"greekrevolution", "aganaktismenoi", "αγανακτισμένοι", "syntagma", "σύνταγμα" ('syntagma' in Greek) and "πραγματική δημοκρατία " ('real democracy' in Greek). The keywords were chosen by virtue of being commonly used on Facebook in the titles and descriptions of pages, and on Twitter as hashtags, to denote content in relation to the mobilizations. Then for every such page, we collect all the publicly available posts using the *fetchConnection* function of RestFB. From the information available we retain, for every post, the Facebook user id of the author of the post and the date of creation of the post (in GMT+2, i.e. local time in Greece). We preemptively performed a few rounds of data collection, in an attempt to mitigate data reliability and validity issues relating to the collection of big data on the web, with the last occurring on January 15th, 2012.

We construct a series of graphs, namely, the *participation graphs*, at different stages. A participation graph is a bipartite graph without multiple edges. One set of vertices corresponds to pages. The other set corresponds to users. There is an edge between a user and a page when the user has contributed at least one post to the page. A participation graph is thus a graph consisting of what can be conceived of as affiliative ties: users loosely affiliated with one another through the process of participating on the same pages. The entire graph contains 43390 vertices (41849 users and 1541 pages) and 72736 edges.

We evaluate several main properties of the graphs that are commonly investigated in literature.

We compute the degree distribution of the graphs. In particular, we compute the degree distribution of the pages and the degree distribution of the users, respectively. In order to show the evolution of the degree distribution, we define four stages, from 10000 users to the maximum number of users, in increments of 10000. Figure D.1 and D.2 show the degree distributions of the pages and the

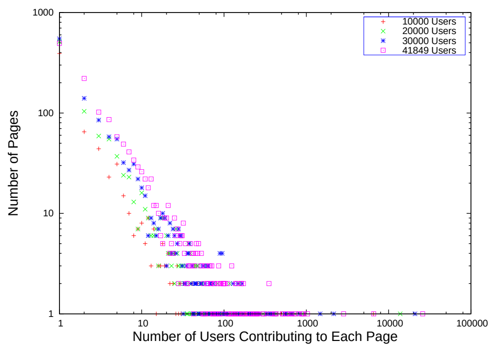


Figure D.1: The distribution of the number of users contributing to each page.

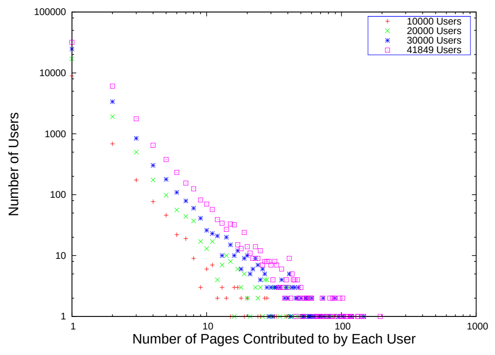


Figure D.2: The distribution of the number of pages contributed to by each user.

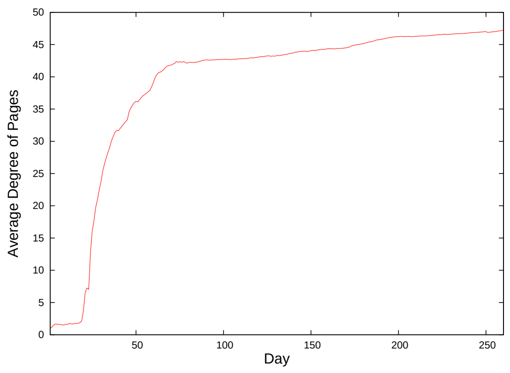


Figure D.3: The evolution of the average degree of pages.

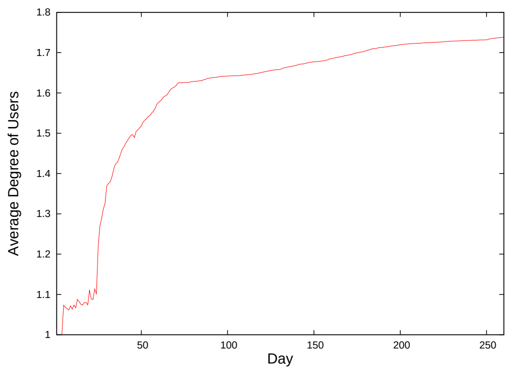


Figure D.4: The evolution of the average degree of users.

users, respectively, in log-scale plot. We observe that both sets of vertices exhibit a heavy-tailed distribution, not only in a single snap shot, but also in the entire history of graph evolution.

We evaluate the evolution of the density of the graph. We compute the daily average degree of pages and users, respectively. Figure D.3 and D.4 show the results. We observe that both sets of vertices exhibit a similar pattern of density evolution. Basically, the density increases in the entire history of graph evolution. The evolution can be further divided into three stages, a slow increasing initial stage, a fast increasing second stage, and a slow increasing final stage.

We evaluate the evolution of the average shortest path length of the graph.

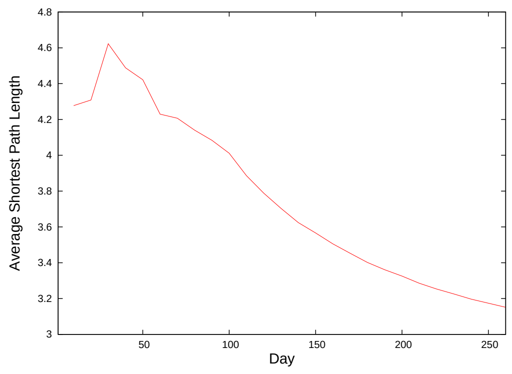


Figure D.5: The evolution of the average shortest path length.

Figure D.5 shows the results. We observe a shrinking average shortest path length during the graph evolution.