Random Walks in the Edge Sampling Model

by

Chang Xu

B.Sc., City University of Hong Kong, 2014

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2016

Approval

Name:	**Chang Xu**
Degree:	**Master of Science (Computing Science)**
Title:	***Random Walks in the Edge Sampling Model***
Examining Committee:	**Chair:** Dr. David Mitchell Associate Professor

Dr. Andrei Bulatov
Co-Senior Supervisor
Professor _____

Dr. Petra Berenbrink
Co-Senior Supervisor
Professor _____

Dr. Leonid Chindelevitch
Internal Examiner
Assistant Professor _____

Date Defended: June 30th, 2016

Abstract

The random walk is an important tool to analyze the structural features of graphs such as the community structure and the PageRank. The problem of generating a random walk may be hard if we are not given full access to the graph. The main component of this thesis is solving the problem in one such model with restricted access to the graph, the edge sampling model. We design Sampling-AS, a randomized algorithm that efficiently samples the endpoint of a random walk, unless some unlikely event happens during the execution of the algorithm. We also propose Sampling-LS, a randomized algorithm that always samples the endpoint of a random walk; however, its performance is not as good. Moreover, we slightly modify both algorithms to improve their performance on some special classes of graphs such as regular graphs, random graphs and fast mixing graphs. Finally, we consider some applications for both algorithms.

Keywords: Random Walk; Edge Sampling Model; Randomized Algorithms

Acknowledgements

First of all, I would like to thank my co-supervisors Dr. Andrei Bulatov and Dr. Petra Berenbrink for their guidance, encouragement and patience. Without their valuable assistance, this thesis would not have been completed. It is my pleasure to be their student.

I would also like to express my appreciation to Dr. Leonid Chindelevitch for kindly reviewing my thesis and Dr. David Mitchell for being the chair of my thesis defence.

Last but not least, I would like to thank my parents and my girlfriend for their love and support during my graduate program. I would also like to thank my friends for proof-reading my thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

As technology and data collection methods develop rapidly, the size of networks such as communication networks, geographic networks and social networks has grown dramatically as well. Approximately 140,000 websites are created in the World Wide Web every day and huge social networks consisting of billions of users and links have been set up by leading Internet companies such as Facebook and Twitter. There is an increasing interest to analyze such networks due to their wide applications. For example, businessmen want to advertise more effectively using social networks; sociologists are interested in finding out communities and elites in networks; epidemiologists are keen to study the spread of infectious diseases within human networks. However, the giant volume of information makes these large networks difficult to analyze.

Although there are many methods to study networks, one of the most successful approaches has been proven to be the method that represents networks by graphs followed by analysing these graphs using graph theory. Researchers define and study various properties of graphs and interpret information related to networks from the topology of the graphs. For instance, Google uses PageRank, a graph property defined by Brin et al. [11] to rate web pages. Another important application is community detection, which assigns vertices into groups so that vertices within the same group are densely connected internally and vertices in different groups are less densely connected. Girvan et al. [21] and many other researchers have tried to detect communities based on graph topology.

A random walk is a stochastic process that starts at an arbitrary node and upon visiting a node v, moves to a randomly chosen neighbor of v. It has been well studied and widely used in algorithms for graphs. For instance, a random walk has been used to construct either a random spanning tree or a random expander graph [12, 30], to solve the load balancing problem [27], to deal with the decentralized routing problem [29], and to model gossip-based communications within networks [28]. In addition, the previously mentioned PageRank and its variation Personalized PageRank [11, 38] make use of the stationary distribution of the random walk.

The random walk is also used in many algorithms for sampling nodes from graphs. Some graphs are too large to be stored in memory and thus one cannot simple sample a node from them. For instance, in the World Wide Web, it is nearly impossible to simple sample a page (node); however, it is possible to see the neighbours of the current page, which enables movement to a random neighbor. Therefore, sampling using random walks is favorable.

Sampling using random walks usually uses the endpoint of a random walk as a sample. Therefore, within the context of this thesis, given a node u and a non-negative integer l, we are interested in the endpoint of a l-length random walk starting at u. Hence, the goal of the thesis is to efficiently sample one node of the network such that the probability that a fixed node v is sampled is the probability that a l-length random walk starting at u ends at v.

Graphs can be represented and accessed in a variety of ways. If a graph is given in an explicit form such as the adjacency matrix or the adjacency list, we would have full access to the graph and performing a random walk on such graphs would be easy and efficient. However, sometimes we only have restricted access to the graph and even accessing the neighbours of a vertex may be hard. This thesis will focus on two models with restricted access to the graph, the *graph streaming model* and the *edge sampling model*.

A good way to process large graphs is by accessing their edges one by one as a stream. The graph streaming model is invented to emulate this process. In this model, a graph is represented by a sequence of edges and it can only be accessed edge by edge. Moreover, normally we only have limited space to process the edge stream since the memory is relatively small compared to the size of the graph.

The edge sampling model is a model that has been used in some studies. In the edge sampling model, there is an oracle which can output an edge per query uniformly at random from a graph. The query is assumed to be expensive and the available space is limited.

The sequence of samples output by the oracle can be viewed as a stream of edges, which looks similar to the graph streaming model. However, streams of edge samples have 2 main differences. First of all, edge samples may contain repeated edges. Secondly, even a large number of edge samples do not guarantee that all edges have been output by the oracle. Thus, we have to use different techniques for these two models.

The edge sampling model also shares some similarities with property testing problems. Similar to many property testing problems, the edge sampling model does not provide us with full information about the graph and it can only be used to infer graph properties or edge weights by sampling (or observation). The lack of full information about the graph will become one of the major obstacles when solving problems within this model. Moreover, we assume that the space is limited and each query is expensive. Therefore, we need to develop algorithms which minimize both the space and the number of queries needed.

In this thesis, we provide efficient algorithms that sample the endpoints of random walks in the graph streaming model and the edge sampling model.

1.1 Related Work

In this section, we briefly review previous works related to random walks, the streaming model and the edge sampling model.

1.1.1 Random Walks

Metropolis designed the famous Metropolis-Hasting algorithm [35] in the 1950s that uses random walks for sampling. The algorithm uses a function proportional to the desired distribution to accept or reject the next possible step of the random walk, so that the distribution of the destination approximates the desired distribution. Sampling using random walks is still a popular research area. Sarma et al. [43] used random walks to estimate PageRank in distributed systems, in which each vertex only knows its own and its neighbours' labels and only $polylog(n)$ bits of data can be sent through one edge per round. Leskovec et al. [31] compared different sampling methods and found that sampling using random walks is among the best methods. Ribeiro et al. [40] was able to use multiple dependent random walks to sample a graph in order to reduce sampling error.

One important result related to this thesis is published in the paper [42] by Sarma et al. who are the first to design semi-streaming algorithms specifically to perform random walks in the graph streaming model. The rough idea of their algorithms is to prepare plenty of short random walks in parallel followed by stitching together these short random walks to reduce the number of passes needed. Their algorithms can perform a random walk[1] using sublinear space and terminate in sublinear number of passes asymptotically. Their algorithms also contain a trade-off between the space and the number of passes needed. Storing many short random walks costs more space but reduces the number of passes.

1.1.2 The Streaming Model

The graph streaming model is a special data streaming model; the latter model has become popular since the publication of Alon et al.'s paper [1] on estimating frequency moments. Frequency moments of a sequence containing m_i elements of type i, for which $1 \leq i \leq n$, are the numbers $F_k = \sum_{i=1}^{n} m_i^k$. Later, Cormode et al. designed a famous count-min sketch which maintains a hashed frequency table to improve frequency estimation [14]. The Lossy Counting algorithm developed by Motwani et al. stores a frequency table and periodically removes items with low frequency [32]. The algorithm is used to identify elements in a data stream whose frequency is greater than a certain threshold. Counting the number of distinct elements is another interesting problem in the streaming model and Kane et al. [24] devised the first optimal algorithm to solve the problem. The algorithm achieves better accuracy by gradually increasing the number of executions of a constant factor approximation algorithm.

[1] Throughout the thesis, performing a random walk means sampling the endpoint of a random walk.

Originally, streaming algorithms only allow a data stream to be examined once. In 2006, Feigenbaum et al. [19] introduced semi-streaming algorithms, which allow the number of passes to be more than one. Using semi-streaming algorithms, more problems can be solved in the streaming model.

There are also many studies related to graph streams. Counting triangles in graph streams were studied both by Sivakumar et al. [4] and Buriol et al. [13]. Sivakumar et al. estimated the number of triangles by reducing the problem to frequency moments estimation in a virtual stream; while Buriol et al. used Reservoir Sampling [45] to uniformly sample an edge and used the probability of the sampled edge belonging to a triangle to estimate the total number of triangles. McGregor [34] designed an algorithm to solve the Maximum Weight Matching problem. The algorithm maintains a matching and it checks whether the weight of a new edge in the graph stream is greater than $1 + \gamma$ times of the total weights of the current matching. If so, the algorithm updates the current matching using the new edge. Guha [22] and McCutchen et al. [33] have studied the K-Center Clustering problem. It involves partitioning input points into K clusters and selecting one center for each part such that the maximum distance between each point within a cluster to its center is minimized. Guha introduced the "streamstrapping" procedure which summarizes part of the initial data to improve the approximation ratio, whereas McCutchen et al. further considered dealing with noise in streams for the K-Center Clustering problem. The problem of constructing a sparse graph spanner has been studied by Elkin [17] and Baswana [6]. Elkin constructed a spanner by accepting vertices with dynamically larger labels, while Baswana constructed a $(2k-1)$-spanner using the idea of finding proper clusters for vertices.

1.1.3 The Edge Sampling Model

Karger showed that uniform edge sampling can be used to generate a graph skeleton which can approximate cut-values under certain conditions [25, 26]. Gao et al. [41] extended Karger's work and proved that the subgraph generated by uniform edge sampling could preserve the original graph's properties like PageRank, thus accelerating community detection by executing algorithms on the sampled subgraph. Ribeiro et al. [40] used uniform edge sampling to estimate degree distributions in social networks.

1.2 Our Contributions

This thesis presents several algorithms that sample the endpoints of random walks in the graph streaming model and the edge sampling model.

In the graph streaming model, we provide a technique to reduce the space required by the SingleRandomWalk algorithm in [42].

For the edge sampling model, we first of all design a randomized algorithm called Sampling-AS. Sampling-AS can be used to sample the endpoint of a l-length random walk

starting at a given node u correctly for both directed and undirected graphs, unless some "bad" event happens (to be explained in Chapter 4). The algorithm uses $\tilde{O}\left(\alpha n + \frac{l^{\frac{1}{2}}}{\alpha}\right)$ space and terminates in $\tilde{O}\left(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha}\right)$ *rounds* (The definition of round will also be explained in Chapter 4.) with high probability[2], where n is the number of vertices in a graph and α is a parameter we can set.

Secondly, for undirected graphs, we design an algorithm called Sampling-LS. Sampling-LS always samples the endpoint of a l-length random walk starting at a given node u. The algorithm uses $\tilde{O}(\alpha n + \alpha l)$ space and terminates in $\tilde{O}\left(l^{\frac{4}{5}} + \frac{l^{\frac{3}{5}}}{\alpha}\right)$ rounds w.h.p.

The two algorithms for the edge sampling model tackle different situations based on the requirements of applications and the type of the graph.

Then in Chapter 5, we analyze the performance of these two algorithms on some special classes of graphs such as regular graphs, random graphs and fast mixing graphs. We find we can achieve better performance by slightly modifying these algorithms.

Finally in Chapter 6, we consider some applications of the Sampling-AS algorithm. First of all, based on the Sampling-AS algorithm, we develop the Sampling-AM algorithm that samples the endpoints of multiple random walks, unless some "bad" event happens (to be explained in Chapter 6). Then we use Sampling-AM to estimate PageRank scores by sampling the endpoints of a large number of random walks. Lastly, we modify Sampling-AM in order to sample the endpoints of absorbing random walks and we use the algorithm in a random walk based recommendation system.

In addition, we notice a possible gap in [42] and we suggest an alternative solution to fix the gap.

[2]Throughout this thesis, an event happens with high probability (shortened to w.h.p.) if the probability that the event happens is $1 - n^{-\Theta(1)}$.

Chapter 2

Preliminaries

This chapter introduces basic terminology and notation for the random walk, the graph streaming model, the edge sampling model and different types of randomized algorithms.

First of all, we give notation used throughout the thesis.

- $G(V, E)$, a graph with vertex set V and edge set E;
- n, the number of vertices in the graph;
- m, the number of edges in the graph;
- l, the length of the random walk to be performed;
- $d_{out}(v)$, the out-degree of vertex v;
- $d_{in}(v)$, the in-degree of vertex v;
- $d(v)$, the degree of vertex v in an undirected graph.

2.1 Random Walk

First, we provide a definition for *stochastic process*.

Definition 2.1.1 (Stochastic Process)**.** *Given a probability space $(\Omega, \mathcal{F}, P)$ (where Ω is the sample space, $\mathcal{F}$ is a σ-algebra of subsets of Ω and P is a countably additive, non-negative measure on $(\Omega, \mathcal{F})$ with total mass $P(\Omega) = 1$) and a measurable space (S, Σ) (where S is the set of values and Σ is a σ-algebra of subsets of S), an S-valued stochastic process is a collection of S-valued random variables on Ω, indexed by a totally ordered set T ("time", usually $\mathbb{N}$). That is, a stochastic process can be defined as a collection*

$$X = \{X_t : t \in T\}, \tag{2.1}$$

where each X_t is an S-valued random variable on Ω. The space S is then called the state space of the process.

The *Markov property* means the future state of the stochastic process depends only on the current state rather than the past states. It is formally defined by the following equation

$$Pr[X_{n+1} = x | X_1 = x_1, X_2 = x_2, ..., X_n = x_n] = Pr[X_{n+1} = x | X_n = x_n]. \tag{2.2}$$

A *Markov chain* is a stochastic process that satisfies the Markov property.

Let the state space of a Markov chain be the vertex set of a graph and make the conditional/transition probability that the state changes from vertex u to v (given the current state is u) as below

$$P(u,v) = \begin{cases} \frac{1}{d_{out}(u)} \text{ , if } (u,v) \in E \\ 0 \text{ , otherwise,} \end{cases} \tag{2.3}$$

then this special Markov chain is called a *random walk* (on directed graphs).

For undirected graphs, the conditional/transition probability that the state changes from u to v (given the current state is u) is

$$P(u,v) = \begin{cases} \frac{1}{d(u)} \text{ , if } \{u,v\} \in E \\ 0 \text{ , otherwise.} \end{cases} \tag{2.4}$$

Therefore, a stochastic process is a random walk if it satisfies the following two conditions.

- ***Condition RW1*** (Markov Property): The stochastic process satisfies the Markov property.
- ***Condition RW2*** (Random Walk Transition): The transition probability between states satisfies the equation (2.3) in directed graphs (or equation (2.4) in undirected graphs).

The initial state X_0 of a random walk is a sampled node from some distribution σ (even if the initial state is a specific vertex u, u can still be considered as a sample from a special distribution σ' where $\sigma'(u) = 1$), and we call σ *initial distribution*.

We let $P_{u,v}$ denote the conditional probability that a random walk moves from vertex u to v, given the walk is currently at u. Then the square matrix P with entries $P_{u,v}$ ($u \in V, v \in V$ assuming vertices are in some order) represents the transition probabilities between each pair of nodes, and it is called the *transition matrix*.

If we let $\pi_\sigma^t(u)$ denote the probability that a random walk is at vertex u after t steps with initial distribution σ, then the vector π_σ^t represents such probabilities for all vertices.

In many cases, this vector approaches some vector after sufficiently many steps and we call this limit vector π^* *stationary distribution*. The stationary distribution satisfies the following equation

$$\pi^* P = \pi^*. \tag{2.5}$$

For undirected graphs, the stationary distribution is easily computed using the equation

$$\pi^*(u) = \frac{d(u)}{2|E|}. \tag{2.6}$$

However, for most directed graphs, finding the stationary distribution is difficult and there are no known formulas.

According to the Fundamental Theorem of Markov Chains [39], every finite, irreducible, and aperiodic Markov chain has a unique stationary distribution. In some problems, the stationary distribution is hard to compute. However, we can use a large number of random walks to estimate the stationary distribution if we can prove its uniqueness.

Mixing time, another important property of a random walk, is used to measure the smallest t needed for π_σ^t to approach the stationary distribution. To understand the definition of mixing time, we give the definition of *total variation distance* first.

Definition 2.1.2 (Total Variation Distance)**.** *Given two distributions μ and π, the total variation distance (TV) is given by*

$$\|\mu - \pi\|_{TV} = \sup_{A \subset V} (\mu(A) - \pi(A)). \tag{2.7}$$

Definition 2.1.3 (Mixing Time [7])**.** *The mixing time $M(\epsilon)$ of a random walk is given by*

$$M(\epsilon) = \max_{\sigma} \min \left\{ t : \left\| \sigma P^t - \pi^* \right\|_{TV} \leq \epsilon \right\}, \tag{2.8}$$

where σ is the initial distribution which converges the slowest to the stationary distribution, π^* is the stationary distribution and P is the transition matrix of the random walk.

Conventionally, the term **random walk** is also used to denote the walk generated by a stochastic process that satisfies Conditions RW1 and RW2. We follow the convention and call such a stochastic process *random walk process* to distinguish the walk generated from the process. In addition, we say multiple random walks are independent if they are generated by independent random walk processes.

Before we end this section, we define *stitching* of random walks and prove an important lemma about it.

Definition 2.1.4 (Stitching)**.** *For two independent random walks A, consisting of a sequence of steps $(a_1, a_2, ...a_m)$ and B, consisting of a sequence of steps $(b_1, b_2, ...b_n)$, given*

*the condition that $a_m = b_1$, the walk C generated by **stitching** A and B is a walk consisting of steps $(a_1, a_2, ...a_m, b_2, b_3, ...b_n)$.*

Lemma 1. *The walk that is generated by stitching a sequence of independent random walks is also a random walk.*

Proof. First of all, we reuse the notation in Definition 2.1.4 and prove that C is a random walk.

We assume A is generated by some stochastic process $\widetilde{A} = (X_1, X_2, ...X_m)$ where the X_is are random variables, and B is generated by some stochastic process $\widetilde{B} = (Y_1, Y_2, ...Y_n)$ where the Y_is are random variables. Then C can be considered to be generated by a stochastic process $\widetilde{C} = (X_1, X_2, ...X_m, Y_2, ...Y_n)$. We denote the event that the state of X_m is the same as that of Y_1 by D. Note that D must happen, otherwise we cannot stitch two random walks.

Since A and B are random walks, $\widetilde{A}$ and $\widetilde{B}$ satisfy Conditions RW1 and RW2. Therefore, the state of each random variable X_i only depends on the state of X_{i-1}. In addition, since A and B are independent, each random variable Y_i's state also only depends on the state of Y_{i-1}. Then we look at X_m and Y_2. The state of Y_2 depends only on the state of Y_1. Under the condition D, the state of Y_2 can be considered as depending only on the state of X_m in the stochastic process $\widetilde{C}$. Therefore, $\widetilde{C}$ satisfies Condition RW1.

Meanwhile, the transition probabilities from X_{i-1} to X_i and those from Y_{i-1} to Y_i satisfy Condition RW2. We also consider X_m and Y_2. For any state y_1 of Y_1 given D happens, and any possible state y_2 of Y_2,

$$Pr[Y_2 = y_2 | (Y_1 = y_1) \cap D] = Pr[Y_2 = y_2 | (X_m = y_1) \cap D]. \tag{2.9}$$

Since the transition probability from Y_1 to Y_2 satisfies Condition RW2, under the condition D, the transition probability from X_m to Y_2 also satisfies Condition RW2.

Therefore, the stochastic process $\widetilde{C}$ satisfies Condition RW1 and Condition RW2. It follows that C is a random walk.

Now we consider stitching a sequence of independent random walks. We assume walk L is generated by stitching a sequence of independent random walks $L_1, L_2, ...L_k$. We can then stitch L_1 and L_2 to $L_{1,2}$ first and use the analysis above to prove that $L_{1,2}$ is a random walk. Next, we can prove $L_{1,2,3}$ that is generated by stitching $L_{1,2}$ and L_3, is a random walk by using the same logic. We inductively stitch the next random walk L_i to previously stitched random walk $L_{1,2...i-1}$ and in the end we prove $L = L_{1,2...k}$ is also a random walk. □

2.2 The Graph Streaming Model

In the graph streaming model, for a graph $G(V, E)$ with vertex set $V = \{v_1, v_2, ..., v_n\}$ and edge set $E = \{e_1, e_2, ..., e_m\}$, a graph stream is a sequence

$$S =< e_{i_1}, e_{i_2}, ...e_{i_m} >, \tag{2.10}$$

where $e_{i_j} \in E$ and $i_1, i_2, ..., i_m$ is an arbitrary permutation of $\{1, 2, ..., n\}$. Each edge of the graph shows up exactly once in the stream and edges may come in any order. We have limited memory to process these edges, normally sublinear in $|V|$. Below is the formal definition of the problem we consider in the graph streaming model.

Problem 1. *Given a graph stream S containing all edges of a connected unweighted directed or undirected graph G in an arbitrary order, with a distribution σ and any $l \geq 0$, sample the endpoint of a l-length random walk with initial distribution σ.*

Note that the problem does not require intermediate steps of the random walk but only the endpoint of a random walk.

We are going to solve the problem using semi-streaming algorithms. Reading through the whole graph stream S once is called one pass and one goal of streaming algorithms is to minimize the number of passes needed.

There are two trivial algorithms to perform a random walk of length l in the graph streaming model.

The first one uses $O(1)$ space to store the current node of the random walk. For each pass, it uses *Reservoir Sampling* (explained in the next section) to uniformly sample an outgoing edge of the current node, and then it extends the random walk by setting the current node to be the other end of the sampled outgoing edge. The total space needed is $O(1)$ and the total number of passes is $O(l)$.

The second algorithm uses one pass and stores all the edges using $O(n^2)$ space. Later on, when the random walk visits a node, we uniformly sample an outgoing edge from all outgoing edges of that node and extend the random walk using the sampled outgoing edge. Therefore, the total space used by the algorithm is $O(n^2)$ and the total number of passes is one.

The two trivial algorithms demonstrate a clear trade-off between the space needed and the number of passes needed.

Reservoir Sampling

Reservoir Sampling, designed by Vitter [45] is a randomized algorithm that uniformly samples an item from a data stream using constant space and one pass. Now we explain how it can be used to uniformly sample outgoing edges of a vertex in the graph streaming model.

We consider a specific vertex v in graph $G(V, E)$, given the graph stream $S =< e_{i_1}, e_{i_2}, ...e_{i_m} >$. We use H to store the edge sample of v, and use c (initially $c = 0$) to count the number of outgoing edges of v that the algorithm has seen so far.

Reservoir Sampling reads edges in S one by one. We skip edges that are not outgoing edges of v. Given an edge e, whose source is v, we increase c by 1 and set H to be e with probability $\frac{1}{c}$. After reading the entire S once, the algorithm outputs H.

We assume v has k outgoing edges and the j-th outgoing edge of v in S is e'. The probability that H is e' by the end of the algorithm is $\frac{1}{j} \cdot (1 - \frac{1}{j+1}) \cdot (1 - \frac{1}{j+2}) ... \cdot (1 - \frac{1}{k}) = \frac{1}{k}$. Therefore, H is a uniform sample from all outgoing edges of v. In addition, the space required is $O(1)$ as the algorithm only stores one edge H and one counter c.

Moreover, if we run s independent instances of Reservoir Sampling for any vertex v, we are able to uniformly and independently sample s edges (with repetition) from all outgoing edges of v using $O(s)$ space and one pass.

2.3 The Edge Sampling Model

The edge sampling model is another model in which we want to sample the endpoints of random walks. We assume there is an oracle which can output an edge uniformly at random from a graph $G(V, E)$ per query. The amount of memory available is limited and the query is expensive. Below we introduce the problem we are going to solve.

Problem 2. *Given an oracle which can return an edge uniformly and randomly from a connected directed or undirected graph G per query, with a distribution σ and any $l \geq 0$, sample the endpoint of a l-length random walk with initial distribution σ.*

Our goal is to solve the problem using as few queries and as little memory as possible.

There is also a trivial algorithm to perform a random walk of length l in the edge sampling model. The algorithm uses $O(1)$ space to store the current node of the random walk and keeps querying the oracle until an outgoing edge of the current node is returned. Then the algorithm will set the current node to be the other end of the sampled outgoing edge. The number of queries needed highly depends on the graph structure. In a d-regular graph, if we denote the number of queries needed for one step by k, then its expectation

$$E[k] = \frac{m}{d} = \frac{d \cdot n/2}{d} = \frac{n}{2}. \tag{2.11}$$

Thus, the total number of queries needed for a random walk of length l is $\frac{nl}{2}$ in expectation.

Unlike the graph streaming model, even after storing a large number of edge samples in the edge sampling model, we cannot guarantee that we have stored all the edges. Therefore, when the random walk visits a node, we cannot select an outgoing edge from all outgoing

edges we have stored for that node to extend the random walk. Without storing any outgoing edge of the node, the walk generated is biased.

2.4 Randomized Algorithms

Randomized algorithms are algorithms whose output or running time are random variables. There are three main categories of randomized algorithms and our algorithms are among two of them. Before introducing these categories, we first give the definition of an algorithm that would correctly solve our problems.

Definition 2.4.1. *We let π_σ^l denote the distribution over vertices such that the probability that a fixed vertex v is sampled is the probability that a random walk with initial distribution σ ends at v after l steps. An algorithm with input σ and l correctly solves our problems if its output X (a random variable) satisfies the condition*

$$\forall u \in V, Pr[X = u] = \pi_\sigma^l(u). \tag{2.12}$$

Definition 2.4.2 (*Las Vegas Algorithm* [3])**.** *An algorithm is a Las Vegas algorithm if its output is always correct but the running time may be unbounded.*

In our problems, the output X of a Las Vegas algorithm with input σ and l satisfies Equation 2.12. In addition, the number of passes/queries and the space required by the algorithm are random variables.

Definition 2.4.3 (*Atlantic City Algorithm* [44])**.** *An algorithm is an Atlantic City algorithm if it is a probabilistic polynomial-time algorithm that answers correctly at least 75% of the time.*

In our problems, the output of an Atlantic City algorithm is correct unless some unlikely event R happens and $Pr[R] < 0.25$. If X is the output of an Atlantic City algorithm,

$$\forall u \in V, Pr[X = u|\overline{R}] = \pi_\sigma^l(u). \tag{2.13}$$

Meanwhile, the number of passes/queries and the space required for the algorithm are random variables.

Definition 2.4.4 (*Monte Carlo Algorithm* [3])**.** *An algorithm is a Monte Carlo algorithm if its running time is deterministic, but its output may be incorrect with a certain (typically small) probability.*

A Monte Carlo algorithms may output incorrect results but its running time is deterministic. Our algorithms do not belong to this category.

Chapter 3

Random Walks in the Graph Streaming Model

This chapter focuses on the connected graphs in the graph streaming model. First, we describe one algorithm from paper [42] by Sarma et al. Then we demonstrate a gap in the analysis given in that paper and suggest an alternative solution to this problem. In the end, we also suggest a technique to reduce the space required for our alternative algorithm.

3.1 Original Algorithm and Analysis

3.1.1 Original Algorithm

Since the paper [42] by Sarma et al. is the starting point for our thesis, it is necessary to begin with its main ideas. We focus on the SingleRandomWalk algorithm in [42] and introduce its ideas here. We also provide the pseudocode of the algorithm for the reader's reference.

The main goal of this algorithm is to sample the endpoint of a random walk with given length l while balancing the space and the number of passes required. Generally speaking, the algorithm consists of two main phases. Phase 1 prepares a large number of w-length short random walks and Phase 2 starts a random walk L_u, tries to use the w-length random walks prepared in Phase 1 to extend L_u and outputs the endpoint of L_u when its length is l.

Phase 1 has two subphases. In Subphase (1.1), the algorithm first samples each node independently with probability α and stores the sampled nodes in a set T (we call vertices in T *connectors* and other nodes *non-connectors*). In Subphase (1.2), the algorithm uses w passes to prepare a w-length random walk for each sampled node in parallel.

In Phase 2, the algorithm samples a node from the given initial distribution and starts L_u at the sampled node. If the random walk visits a connector for the first time, the random walk can take advantage of the prepared w-length random walk and proceed w steps without

reading the stream. One important point is that the same w-length random walk cannot be used more than once, otherwise this stochastic process's future state would also depend on the past states thus violating the Markov property. Therefore, the algorithm uses a set S to store connectors whose w-length random walks have been used (we call these connectors *used connectors* and other connectors *unused connectors*). If a connector has been added into S, then its w-length random walk cannot be used again.

In Phase 2, L_u might also be ***stuck*** on either a used connector or a non-connector. Being stuck means that L_u visits a node without an available w-length random walk and we call such a node *stuck node*. The algorithm proposes a subroutine HandleStuckNode to deal with this scenario. The subroutine uses a set R to store any non-connector the random walk visits during the current invocation of the subroutine. Then the subroutine uniformly samples s outgoing edges for each node in $S \cup R$ using one pass (we denote this procedure as Subphase (2.1)). Afterwards, L_u would use the sampled s edges when it visits a node in $S \cup R$. In the case where the random walk quickly leaves the set $S \cup R$, the new node it visits is an unused connector with probability α and L_u can make αw progress[3]in expectation. If L_u keeps visiting vertices in $S \cup R$, when it visits a vertex $v \in S \cup R$ $s+1$ times, v does not have edge samples to extend L_u. Then, we say L_u is ***trapped*** and we need to use an additional pass to sample s edge samples for vertices in $S \cup R$. Even if the random walk is trapped, L_u can still make at least s progress after one pass since L_u uses up all s edges of v.

This technique helps extend the random walk efficiently. In some graphs, L_u might easily leave $S \cup R$ and frequently visit unused connectors. Then L_u is likely to use many prepared w-length random walks to proceed quickly. In graphs where L_u is trapped within a small group of nodes and unable to visit many unused connectors, it can still use the subroutine HandleStuckNode to proceed with at least s steps after each pass.

Below is the SingleRandomWalk algorithm and its subroutine HandleStuckNode from paper [42].

3.1.2 Original Analysis

In this section, we will briefly go over the analysis in the paper [42] since it is relevant to our study. The core result is the following theorem.

Theorem 1 (Theorem 3.1 in [42])**.** *For any graph G with n vertices and any $l \geq 0$, the endpoint of a random walk of length l can be sampled using $\tilde{O}(\sqrt{\frac{l}{\alpha}})$ passes and $\tilde{O}(\alpha n + \sqrt{\frac{l}{\alpha}} + \alpha l)$ space w.h.p., for any choice of α with $0 < \alpha \leq 1$.*

First of all, Sarma et al. [42] prove the correctness of the SingleRandomWalk. During the entire algorithm, no w-length random walk is reused. In addition, in each invocation of

[3]A random walk makes x progress meaning the random walk proceeds with x steps.

Algorithm 1 SingleRandomWalk(u, l)

Input: Starting node u and desired walk length l.
Output: endpoint of L_u (the random walk from u of length l)
Phase 1 starts
Subphase (1.1) $T \leftarrow$ set of nodes obtained by sampling each node independently with probability α (in one pass and we refer to nodes in T as connectors).
Subphase (1.2) In w passes, perform walks of length w from every node in T. Let $W[t] \leftarrow$ the end point of the walk of length w from $t \in T$ (the nodes in T whose w length walks get used by L_u will get included in S).
Phase 2 starts
Initialize L_u to be a zero length walk starting at u and let $x \leftarrow u$
$S \leftarrow \{\ \}$ (which is used to store used connectors)
while $|L_u| < l$ **do**
    if $x \in T$ and $x \notin S$ **then**
        Extend L_u by appending the available w-length walk (implicitly by setting the current endpoint x of L_u to be the endpoint of the w-length walk starting from x). $S \leftarrow S \cup \{x\}$, and let $x \leftarrow W[x]$ { This means we have a w-length walk starting at x that has not been used so far in L_u. }
    else
        HandleStuckNode(x, T, S, L_u, l) { This means L_u either reaches an non-connector or a used connector, then we will use subprocess HandleStuckNode to deal with it.)
    end if
end while
return the endpoint of L_u

the HandleStuckNode, since L_u uses the k-th edge sample while visiting a node for the k-th time, s sampled edges are only used at most once. Therefore, each future state does not depend on the past states and this stochastic process satisfies Condition RW1. Moreover, if the current state is u, the process uniformly and randomly picks a neighbor of u to be the next state. Hence, this Markov chain also satisfies Condition RW2. Therefore, the generated L_u is a random walk.

Secondly, paper [42] analyzes the space complexity of the algorithm. The paper claims the space required to store the endpoints of all w-length random walks for connectors is $O(\alpha n)$. Each connector only requires $O(1)$ space to store one endpoint and there are $O(\alpha n)$ connectors. ($O(\alpha n)$ should be a typo since the number of connectors can only be bounded by $\tilde{O}(\alpha n)$, which will be explained later.) During Phase 2, space is needed for set S and sets R. Regarding the bound of the space required for S and R, paper [42] proves the following lemma and claim.

Lemma 2 (Lemma 3.4 in [42])**.** $|S| \leq l/w$

Proof. A node is added to the set S only after we use a w-length walk from one of the connectors. If we perform a walk of length l, we will end up using at most l/w walks of length w. □

Algorithm 2 HandleStuckNode(x, T, S, L_u, l)

$R \leftarrow x$
while $|L_u| < l$ **do**
    Subphase (2.1) $E_u \leftarrow$ sample s edges (with repetition) for every node $u \in S$ (using one pass)
    Extend L_u using edges sampled in Subphase (2.1). (When visiting a node $v \in S$ for the k-th time, use the k-th sampled edge from E_v.)
    $x \leftarrow$ new endpoint after extending L_u (Then there are three cases.)
    (1) if ($x \in S \cup R$), **continue**[4]{ no new node is seen, at least s progress is made.}
    (2) if ($x \in T$ and $x \notin S \cup R$), **return**[5]{this means that x is a node that has not been seen in the walk so far, and x is a connector; therefore, the w-length walk from x has not been used}
    (3) if ($x \notin T$ and $x \notin S \cup R$), $R \leftarrow R \cup \{x\}$, **continue** {this means that x is a new node that has not been visited in this invocation, and x is not in the initial set of sampled nodes T }
end while
return the endpoint of L_u

Claim 1 (Claim 3.5 in [42])**.** *With every additional pass over the edge stream (after the first w passes), the length of the random walk L_u increases by at least $O(min\{s, \alpha w\})$ in amortization.*

Proof. We only need to examine the algorithm HandleStuckNode. An additional pass over the stream is made when s edges are sampled from every node in $S \cup R$. This happens when the algorithm gets stuck at a new stuck node in R. After a pass over the stream, either the algorithm makes s progress, or a *new node* is visited (where *new node* stands for a node that is not in $S \cup R$). In the latter case, with probability α, the new node is in T (since T contains each node with probability α), and with probability $1 - \alpha$, it is a new stuck node. If the new node is not a stuck node, w progress is made. Since the probability of not seeing a new node in T is $1 - \alpha$ with every additional pass, the probability that more than $\tilde{O}(1/\alpha)$ new stuck nodes are seen before a new node in T is seen to be small by Chernoff bound. Therefore, w.h.p., $|R|$ is less than $\tilde{O}(1/\alpha)$ in each invocation of HandleStuckNode. Hence, the number of passes in which s progress is not made, is no more than $\tilde{O}(1/\alpha)$, at the end of which w progress is made giving $\frac{w}{\tilde{O}(1/\alpha)}$ average progress per pass. By this amortized argument, the walk makes $O(min\{s, \alpha w\})$ progress with every pass over the edge stream. Furthermore, w.h.p., in r passes, the algorithm SingleRandomWalk makes a progress of at least $O(r \cdot min\{s, \alpha w\})$ steps (for $r \geq \Omega(1/\alpha)$). □

Using the above lemma and claim, [42] proves that the size of $S \cup R$ is $\tilde{O}(\frac{l}{w} + \frac{1}{\alpha})$ w.h.p. Because each node in $S \cup R$ stores s edge samples in the HandleStuckNode, the space complexity for the entire algorithm is $\tilde{O}(\alpha n + s(\frac{l}{w} + \frac{1}{\alpha}))$ w.h.p.

[4]Throughout the thesis, **continue** means that the described algorithm starts another **while** loop.
[5]Throughout the thesis, **return** means that the described algorithm exits the current subroutine.

Then paper [42] bounds the number of passes needed. In Phase 1, the algorithm needs $w+1$ passes to sample nodes in T and it generates a w-length random walk for each connector. In Phase 2, from Claim 1, the total number of passes is $\frac{l}{O(min\{s,\alpha w\})}$ w.h.p. Thus, the total number of passes is $\tilde{O}(w+\frac{l}{s}+\frac{l}{\alpha w})$ w.h.p.

After setting $s=\sqrt{l\alpha}$ and $w=\sqrt{\frac{l}{\alpha}}$, Theorem 1 is proved.

3.2 The Gap and Our Solution

In the beginning, we introduce the term *stop*. The random walk L_u stops at a node when this node is assigned to x in the pseudocode. If a node is visited by the random walk within the w-length short random walks, the node is not considered to be stopped at by the random walk.

3.2.1 The Gap

The gap comes from the proof of Claim 1. In the original analysis for the subroutine HandleStuckNode, after an execution of Subphase (2.1), the random walk L_u will use the sampled edges to proceed and we end up with one of the following cases. (Let x be the endpoint of L_u at this time.)

Case 1. After the current execution of Subphase (2.1), L_u does not leave $S\cup R$ and is trapped, that is $x\in S\cup R$.

Case 2. L_u leaves $S\cup R$, that is $x\notin S\cup R$. (This is the case that random walk stops at a new node, which has the two subcases below.)

Case 2(a) The new node is an unused connector, $x\in T$ and $x\notin S\cup R$.

Case 2(b) The new node is a non-connector, $x\notin T$ and $x\notin S\cup R$.

If Case 1 happens, the random walk makes at least s progress. If Case 2 happens, [42] says that with probability α Case 2(a) happens, and with probability $1-\alpha$ Case 2(b) happens. To be more precise, we denote Y to be the event that the random walk leaves $S\cup R$ (Case 2) and let X denote the event that x is a connector ($X\cap Y$ is Case 2(a)). Then, [42] claims that

$$Pr[X|Y]=\alpha. \tag{3.1}$$

However, we argue that for the original SingleRandomWalk, it is possible that

$$Pr[X|Y]<\alpha. \tag{3.2}$$

To prove this, we first need the following observation.

Observation 1. *In the SingleRandomWalk algorithm, if the random walk stops at a node for the first time, the probability that the node is a connector is α.*

Proof. We let Z denote the event that the random walk stops at some node u for the first time. X denotes the same event as defined above. Thus, we claim

$$Pr[X|Z] = \alpha. \tag{3.3}$$

To prove this equation, we use the following arguments. Given the condition that the random walk stops at some node u for the first time, the event that u is a connector is the same event that u is sampled as a connector in Subphase (1.1). Therefore, this conditional probability should also be α. More importantly, such conditional probabilities for every node are independent since we sample every node as connector independently.

□

Now, we prove Inequality (3.2).

Proof. We reuse the same events X and Y. In event Y, x is outside of $S \cup R$ and x can be in two mutually exclusive events.

1. The random walk stops at x for the first time. (Denote this event as Y_1.)
2. The random walk has stopped at x before. (Denote this event as Y_2.)

The reason Y_2 might happen is because the algorithm does not store all the vertices that L_u has stopped at. Although S stores all used connectors, R only stores non-connectors that L_u has stopped at in the current invocation of the HandleStuckNode. Consider a non-connector v that the random walk has stopped at in previous invocations of the HandleStuckNode. If L_u happens to stop at v, the event Y_2 happens. Therefore, $Pr[Y_2] > 0$.

According to the definition of conditional probability,

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}. \tag{3.4}$$

Since the algorithm adds used connectors to S, if the event Y_2 happens, L_u will only stop at a non-connector that has been stopped at before, which means

$$Pr[X|Y_2] = 0 \text{ and } Pr[X \cap Y_2] = 0. \tag{3.5}$$

Therefore, due to Equation (3.5),

$$Pr[X|Y] = \frac{Pr[X \cap Y_1] + Pr[X \cap Y_2]}{Pr[Y]} = \frac{Pr[X \cap Y_1]}{Pr[Y]}. \tag{3.6}$$

Since $Y_1 \cup Y_2 = Y$ and $Y_1 \cap Y_2 = \emptyset$,

$$Pr[X|Y] = \frac{Pr[X \cap Y_1]}{Pr[Y_1] + Pr[Y_2]} < \frac{Pr[X \cap Y_1]}{Pr[Y_1]} = Pr[X|Y_1]. \tag{3.7}$$

By Observation 1,

$$Pr[X|Y] < Pr[X|Y_1] = \alpha. \tag{3.8}$$

□

In the proof of Claim 1, [42] uses Equation (3.1) to prove the space bound for R and also the bound for the number of passes. Since this equation is not correct, the original bounds do not hold any more.

To illustrate the gap clearly, we include the following five figures to show a situation where the gap occurs. Graph G is a connected graph and the edges are not shown. White points represent the vertices that the random walk L_u has never stopped at. Black points represent the non-connectors that L_u has stopped at, while red or gray points represent the connectors that L_u has stopped at. In addition, straight arrow means L_u proceeds with one regular random walk step while curved arrow means L_u uses a prepared w-length random walk.

Figure 3.1 is the state of the graph after connectors are sampled and w-length random walks are prepared. L_u starts at vertex u. Since no vertices have been stopped at, all of them are white and with probability α they are connectors by Observation 1. In addition, R and S are empty.

Initially, the random walk stops at u and with probability $1-\alpha$, u happens to be a non-connector. Thus, the algorithm uses the subroutine HandleStuckNode and stores s into R. Assume L_u visits a and b in the next two steps and they both happen to be non-connectors, then a and b are added to R as well. Figure 3.2 illustrates the state where R is $\{u, a, b\}$ and S is still empty.

In the next step, L_u visits c and with probability α, c happens to be a connector. Thus, this invocation of the HandleStuckNode completes and the space for current R will be released. This state is shown in Figure 3.3.

In Figure 3.4, L_u utilizes the w-length random walk of c and adds c into S. Let us assume that L_u comes back to the vertex u. Since u is a non-connector, L_u is stuck again and the algorithm starts another invocation of HandleStuckNode.

Finally in Figure 3.5, we assume L_u visits a again (the step is represented by the bold black arrow). Since $a \notin R \cup S$, a is considered to be a *new* node in the original analysis. The original analysis argues that a is a connector with probability α at this step. The argument is incorrect since a is a non-connector for sure.

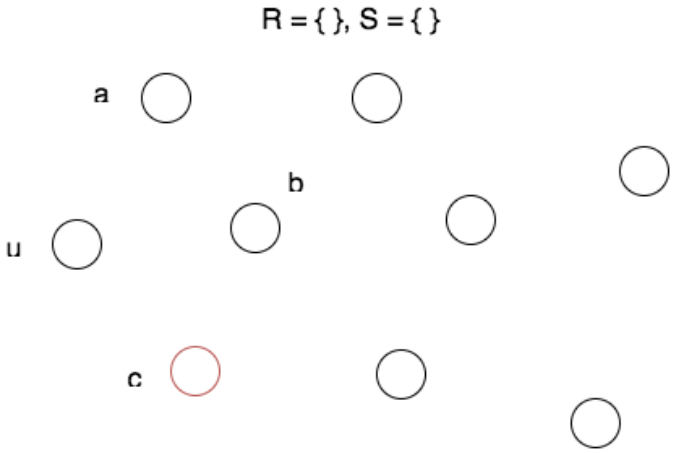

Figure 3.1: The initial state of the graph of Phase 2

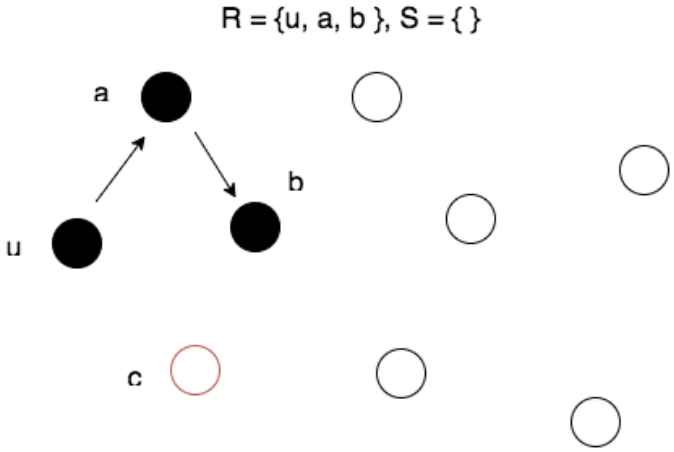

Figure 3.2: The random walk L_u has stopped at some non-connectors

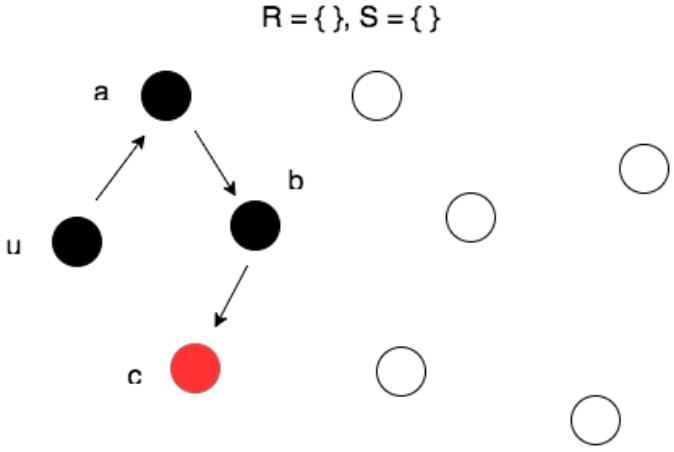

Figure 3.3: L_u stops at a connector

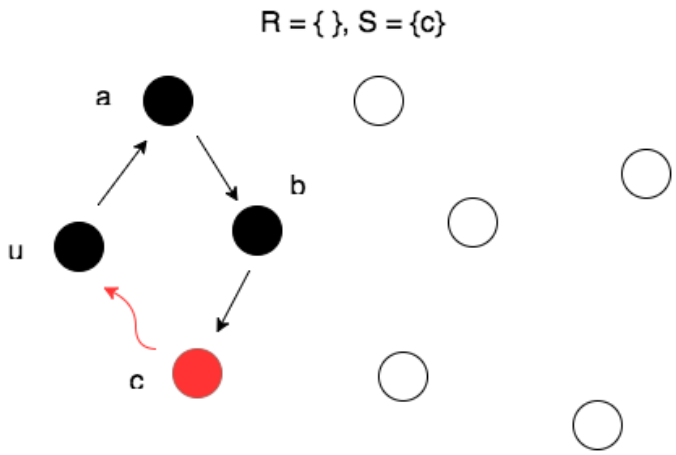

Figure 3.4: L_u utilizes the w-length random walk of c and comes back to u

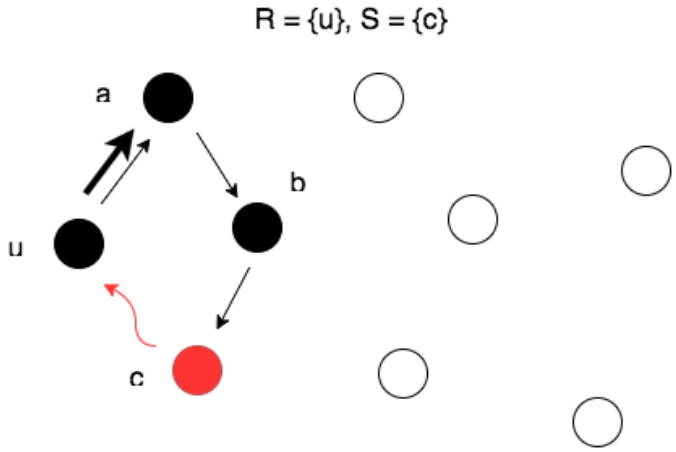

Figure 3.5: L_u stops at a previous non-connector again

3.2.2 Our Alternative Solution

We propose our alternative solution to avoid the gap described above. Our algorithm discards the set R and uses the set S to store all the vertices that the random walk has stopped at (no matter whether they are connectors or not). For a node u, if it is not stored in S, then it is a node that the random walk has never stopped at. With probability α, u is a connector by Observation 1. The algorithm using our alternative solution is called *Streaming-S* (short for Streaming Single Random Walk) whose subroutine is called *Streaming-H*. The following pseudocode provides the details.

Algorithm 3 Streaming-S(u, l)

Input: Starting node u and desired walk length l.
Output: The endpoint of L_u (the random walk from u of length l)
Phase 1 starts
Subphase (1.1) $T \leftarrow$ set of nodes obtained by sampling each node independently with probability α (in one pass and we refer to nodes in T as connectors).
Subphase (1.2) In w passes, perform walks of length w from every node in T. Let $W[t] \leftarrow$ the end point of the walk of length w from $t \in T$
Phase 2 starts
Initialize L_u to be a zero length walk starting at u and let $x \leftarrow u$
$S \leftarrow \{\ \}$ (which is used to store all vertices the random walk has stopped at)
while $|L_u| < l$ **do**
    if $x \in T$ and $x \notin S$ **then**
        Extend L_u by appending the available w-length walk (implicitly by setting the current endpoint x of L_u to be the endpoint of the w-length walk starting from x). $S \leftarrow S \cup \{x\}$, and let $x \leftarrow W[x]$ { This means we have a w-length walk starting at x that has not been used so far in L_u. }
    else
        Streaming-H(x, T, S, L_u, l) { This means L_u either reaches a non-connector or a used connector, then we will use subroutine Streaming-H to deal with it.)
    end if
end while
return the endpoint of L_u

The Streaming-S algorithm stores much more vertices than the original SingleRandomWalk algorithm. Therefore, our algorithm has a different probabilistic bounds on the space and number of passes needed, and the following theorem states the new bounds. We will prove it in Sections (a) and (b) below.

Theorem 2. *For any connected graph G with n vertices and any integer $l \geq 0$, the Streaming-S algorithm always samples the endpoint of a l-length random walk. It uses $\tilde{O}(\alpha n + \frac{l^{\frac{1}{2}}}{\alpha})$ space and terminates in $\tilde{O}(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha})$ passes w.h.p. In particular, if $l = O(n)$ and $\alpha = n^{-\frac{1}{4}}$, the space complexity of Streaming-S is $\tilde{O}(n^{\frac{3}{4}})$ and the number of passes required is $\tilde{O}(n^{\frac{3}{4}})$ w.h.p.*

Algorithm 4 Streaming-H(x, T, S, L_u, l)

while $|L_u| < l$ **do**
  Subphase (2.1) $E_u \leftarrow$ sample s edges (with repetition) for every node $u \in S$. (using one pass)
  Extend L_u using edges sampled in Subphase (2.1). (When visiting a node $v \in S$ for the k-th time, use the k-th sampled edge from E_v.)
  $x \leftarrow$ new end point after extending L_u (Then there are three cases.)
  (1) if $(x \in S)$, **continue** { no new node is seen, at least s progress is made.}
  (2) if ($x \in T$ and $x \notin S$), **return** {this means that x is a node that has not been stopped at by L_u so far, and x is a connector; therefore, the w-length walk from x has not been used}
  (3) if ($x \notin T$ and $x \notin S$), $S \leftarrow S \cup \{x\}$, **continue** {this means that x is a node that has not been stopped at by L_u, and x is not a connector }
end while
return the endpoint of L_u

(a) Space

First of all, we need the following result about the number of connectors.

Lemma 3. *W.h.p., the number of connectors is $\tilde{O}(\alpha n)$.*

Proof. For every node $u \in V$, we define X_u to be 1, if u is sampled to be a connector and 0 otherwise. Let $X = \sum_{u \in V} X_u$, then $E[X] = \alpha n$. As the X_us are independent, identical and 0-1 random variables, Chernoff bound can be applied to X to derive that

$$Pr[X > (1+\delta) \cdot E[X]] \leq e^{-\frac{\delta^2}{2+\delta} E[X]}. \tag{3.9}$$

Let $\delta = 2\log(n)$ and, since $\log(n)$ can be assumed to be greater than 2,

$$Pr[X > (1 + 2\log(n)) \cdot \alpha n] \leq e^{-\log(n) \cdot \alpha n} = n^{-\alpha n}. \tag{3.10}$$

We assume $\alpha n \geq 1$ (otherwise there are almost no connectors) and denote the event that the number of connectors is not $\tilde{O}(\alpha n)$ by Q_1. Then, $Pr[Q_1] \leq \frac{1}{n}$.

□

From Lemma 3, we know there are $\tilde{O}(\alpha n)$ connectors w.h.p. Since we only store the endpoint of each w-length random walk, we only need $O(1)$ space for each connector. Therefore, w.h.p. the total space required for storing connectors and generating w length random walks is $\tilde{O}(\alpha n)$.

In order to bound the space complexity for S, we need the following lemma.

Lemma 4. *W.h.p., for the sequence of nodes added to S, there are at most $\tilde{O}(\frac{1}{\alpha})$ consecutive non-connectors before a connector occurs through the algorithm.*

Proof. In the beginning, we uniformly sample connectors with probability α. Therefore, the conditional probability that a node u is a connector equals α by Observation 1, conditioned on the event that u is a node that the random walk has never stopped at. If a node is to be added to S, the random walk must stop at the node for the first time. Thus, for every new node added to S, the node is a connector with probability α.

We set P_i to be the probability that i consecutive non-connectors are added to S, then

$$P_i = (1-\alpha)^i. \tag{3.11}$$

In addition, since $1-\alpha < e^{-\alpha}$,

$$\log\left(\frac{1}{1-\alpha}\right) = -\log(1-\alpha) > \alpha. \tag{3.12}$$

If we set $i = \frac{\log(l\cdot n)}{\alpha}$, then by Inequality (3.12),

$$i = \frac{\log(l\cdot n)}{\alpha} > \frac{\log(l\cdot n)}{\log\left(\frac{1}{1-\alpha}\right)}. \tag{3.13}$$

Therefore,

$$P_i = (1-\alpha)^{\frac{\log(l\cdot n)}{\alpha}} < (1-\alpha)^{\frac{\log(l\cdot n)}{\log\left(\frac{1}{1-\alpha}\right)}} = \frac{1}{nl}. \tag{3.14}$$

Hence, for a specific connector v in S, the probability that there are $\tilde{O}(\frac{1}{\alpha})$ consecutive non-connectors added before v is at most $\frac{1}{nl}$. Let Q_2 denote the event that there exists some connector $u \in S$, for which there are more than $\tilde{O}(\frac{1}{\alpha})$ consecutive non-connectors added before u. By the union bound, $Pr[Q_2] \leq \frac{1}{nl}\cdot|S| \leq \frac{1}{n}$. □

The maximum number of connectors that may be used is $\frac{l}{w}$ by Lemma 2. By Lemma 4, w.h.p. $|S|$ is at most $\tilde{O}(\frac{l}{\alpha w})$. Since each node in S stores s edge samples, the total space required by the Streaming-H is $\tilde{O}(\frac{ls}{\alpha w})$ w.h.p.

We denote the union of events Q_1 and Q_2 as Q. By the union bound, $Pr[Q] \leq \frac{2}{n}$. Therefore, the total space needed for the Streaming-S algorithm is $\tilde{O}(\alpha n + \frac{ls}{\alpha w})$ w.h.p.

If $l = O(n), \alpha = n^{-\frac{1}{4}}, s = n^{\frac{1}{4}}, w = n^{\frac{3}{4}}$, the space required is $\tilde{O}(n^{\frac{3}{4}})$ w.h.p. Compared to the algorithm in [42], although the space complexity increases from $\tilde{O}(n^{\frac{2}{3}})$ to $\tilde{O}(n^{\frac{3}{4}})$, it is still sublinear.

(b) Number of Passes

First of all, sampling connectors uses one pass and generating w-length random walks for connectors uses w passes.

Since the random walk does not need additional passes to utilize w-length random walks, we only need to bound the number of passes required by Subphase (2.1). There are two cases where we need to use additional 1 pass to sample s edge samples for nodes in S.

Case 1. The random walk L_u stops at a stuck node $x \notin T$ and $x \notin S$. (This is the case where x is a new node for S and is not a connector, then we will add it to S.)

Case 2. L_u does not leave S and is trapped at a node $x \in S$.

In both cases, we can use one pass to draw s samples for each node in S. The number of times Case 1 occurs is bounded by $|S|$, which is $\tilde{O}(\frac{l}{\alpha w})$ w.h.p from the analysis for the space complexity. The number of times Case 2 occurs is bounded by $O(\frac{l}{s})$, since each time Case 2 occurs, the random walk will make at least s progress. Thus, the total number of passes from both cases is $\tilde{O}(\frac{l}{\alpha w} + \frac{l}{s})$ w.h.p.

From the analysis above, the total number of passes required by the Streaming-S algorithm is $\tilde{O}(w + \frac{l}{\alpha w} + \frac{l}{s})$ w.h.p.

If $l = O(n), \alpha = n^{-\frac{1}{4}}, s = n^{\frac{1}{4}}, w = n^{\frac{3}{4}}$, the total number of passes is $\tilde{O}(n^{\frac{3}{4}})$. The number of passes increases from $\tilde{O}(n^{\frac{2}{3}})$ to $\tilde{O}(n^{\frac{3}{4}})$ compared to the algorithm in [42], but it is still sublinear.

With the help of the previous analysis for space complexity and the number of passes, if we let $w = l^{\frac{3}{4}}$ and $s = l^{\frac{1}{4}}$, Theorem 2 is obtained.

3.2.3 Space Improvement

Since the Streaming-S algorithm stores all vertices that the random walk has stopped at, the size of set S grows quickly. As a result, storing s edges for each node in S uses a large amount of space.

The space required by the algorithm can be reduced in some graphs using the following technique. In the subroutine Streaming-H, previously we store s edges for each node in S, which takes $O(|S|s)$ space. s edges are required for each vertex, since even if the random walk L_u keeps visiting the vertex, L_u can still make at least s progress by using the sampled s edges. However, if the number of outgoing edges of a node is smaller than s, we only need to store all outgoing edges of the node using space less than s. When L_u visits the node, we uniformly sample an outgoing edge from all outgoing edges stored, and use this edge to extend L_u. After storing all outgoing edges of a vertex, sufficient amount of edge samples can be generated no matter how many times the random walk visits the node. Moreover, in order to identify the vertices with out-degree no more than s, we need an additional pass to count the out-degree for each vertex in S.

Using the idea above, we design a new subroutine Streaming-SH (short for Streaming Space Saving HandleStuckNode) below. We propose a new algorithm Streaming-SS by

replacing Streaming-H with Streaming-SH. The Streaming-SS algorithm can be used if we know in advance that only few nodes have a large number of outgoing edges.

Algorithm 5 Streaming-SH(x, T, S, L_u, l)

while $|L_u| < l$ **do**
    Subphase (2.1) Count the out-degree for each vertices in S and partition S into two subsets F and M. F contains every node whose out-degree is no more than s, while M contains every node whose out-degree is larger than s. (using one pass)
    For each node u in F, store all distinct outgoing edges of u into E_u. For each node v in M, uniformly sample s outgoing edges (with repetition) into E_v for v. (using one pass)
    Extend L_u using previously stored edges. (When visiting a node $u \in F$, use an edge sampled uniformly from E_u; when visiting a node $v \in M$ for the k-th time, use the k-th sampled edge E_v.)
    $x \leftarrow$ new end point after extending L_u (Then there are three cases.)
    (1) if $(x \in S)$, **continue** { no new node is seen, at least s progress is made.}
    (2) if $(x \in T$ and $x \notin S)$, **return** {this means that x is a node that has not been stopped at by L_u so far, and x is a connector; therefore, the w-length walk from x has not been used}
    (3) if $(x \notin T$ and $x \notin S)$, $S \leftarrow S \cup \{x\}$, **continue** {this means that x is a new node that has not been stopped at by L_u, and x is not a connector }
end while
return the endpoint of L_u

The number of passes for the Streaming-SS algorithm is increased by a constant factor of at most two since Subphase (2.1) uses an additional pass. The Streaming-SS algorithm needs extra $O(|S|)$ space when counting the out-degree of each vertex in S and partitioning these vertices, but it saves the space that stores edges for vertices in S. The space improvement effect of the algorithm depends highly on the graph structure, especially on the degree distribution. In Chapter 5, we will show that the algorithm performs well in special classes of graphs like regular graphs.

Chapter 4

Random Walks in the Edge Sampling Model

In this chapter, we use an idea similar to that in the Streaming-S algorithm to sample the endpoints of random walks on connected graphs in the edge sampling model. As before, we sample connectors and prepare a w-length random walk for each connector in parallel. Then, we make use of the prepared w-length random walks to quickly extend a random walk L_u.

As mentioned before, for any vertex u in the graph streaming model, we can use one pass to uniformly sample any amount of outgoing edges of u by running multiple instances of the Reservoir Sampling algorithm in parallel.

However, the Reservoir Sampling algorithm cannot be applied in the edge sampling model since the oracle might output repeated edges. In the edge sampling model, the oracle returns an edge uniformly at random. Therefore, for a vertex v of low degree, the oracle is unlikely to output any outgoing edge of v. Because Subphase (1.2) and the subroutine Streaming-H both need many edge samples for vertices of low degree in the graph streaming model, we have to query the oracle many times to get a sufficient number of edge samples in the edge sampling model. In order to reduce the number of queries, we need special strategies to deal with these bottlenecks.

Based on different strategies we use, we will give one Atlantic City algorithm that could be used for directed or undirected graphs. We also give a Las Vegas algorithm for undirected graphs. The Las Vegas algorithm uses slightly more queries and space, but its output is guaranteed to be correct.

Round

In order to better compare algorithms in the edge sampling model, we define one round to be a sequence of $m \log(mnl)$ queries so that after one round all edges will be returned

by the oracle at least once w.h.p. The following lemma follows from the coupon collector problem.

Lemma 5. *The probability that some edge is not sampled in k rounds is at most $n^{-k}l^{-1}$.*

Proof. We use $P_i(x)$ to represent the probability that the i-th edge is not sampled after x queries (we assume edges have some order),

$$P_i(x) = \left(\frac{m-1}{m}\right)^x = \left(1 - \frac{1}{m}\right)^{mx\frac{1}{m}} < e^{-\frac{x}{m}}. \tag{4.1}$$

Let $P(x)$ denote the probability that at least one edge is not sampled after x queries. By the union bound,

$$P(x) \leq \sum_{i=1}^{m} P_i(x) = m \cdot P_i(x) < m \cdot e^{-\frac{x}{m}}. \tag{4.2}$$

For k rounds, x equals to $km\log(mnl)$,

$$P(km\log(mnl)) < m \cdot e^{-k\log(mnl)} = \frac{1}{m^{k-1}n^k l^k} \leq n^{-k}l^{-1}. \tag{4.3}$$

Therefore, all edges will be sampled at least once in k rounds w.h.p. □

4.1 The Atlantic City Algorithm

First of all, we introduce the Atlantic City algorithm. We name this algorithm Sampling-AS (short for Sampling Atlantic City Single Random Walk) and its subroutine Sampling-AH. The Sampling-AS algorithm always samples the endpoint of a random walk with length l, unless some unlikely event happens.

Since the Sampling-AS algorithm has a very similar structure to the Streaming-S algorithm, we reuse the terminology (connectors, non-connectors et al.) and notation (S, R, L_u et al.) from the Streaming-S algorithm.

In Phase 1, the Sampling-AS algorithm samples connectors in Subphase (1.1) and prepares one w-length walk for each connector using $2w$ rounds in Subphase (1.2).

In Phase 2, when L_u (the walk we generate) stops at connectors, the Sampling-AS algorithm uses the prepared w-length walks to extend L_u. When L_u stops at a stuck node, the subroutine Sampling-AH is used. It samples s edges for all stuck nodes in S (S contains all vertices that L_u has stopped at) using 2 rounds, and extends L_u using the sampled edges.

The pseudocode is presented below to give a general idea of the algorithm. The key steps will be explained in Section 4.1.1.

L_u constructed by the Sampling-AS algorithm might violate Conditions RW1 and RW2 of a random walk if some "bad" events happen. The union of all "bad" events is denoted by R and will be explained in detail in Section 4.1.2.

Algorithm 6 Sampling-AS(u, l)

Input: Starting node u and desired walk length l.
Output: The endpoint of L_u (the walk starting from u of length l)
Phase 1 starts
Subphase (1.1) Sample each node appearing in a round independently with probability α, and store the sampled nodes into a set T
Subphase (1.2) In $2w$ rounds, perform a w-length walk from every node in T. Let $W[t]$ be the end point of the w-length walk from $t \in T$
Phase 2 starts
Initialize L_u to be a zero length walk starting at u and let $x \leftarrow u$
$S \leftarrow \{\ \}$ (which is used to store vertices that L_u has stopped at)
while $|L_u| < l$ **do**
    if $x \in T$ and $x \notin S$ **then**
        Extend L_u by appending the available w-length walk (implicitly by setting the current endpoint x of L_u to be the endpoint of the w-length walk starting from x). $S \leftarrow S \cup \{x\}$, and let $x \leftarrow W[x]$.
    else
        Sampling-AH(x, T, S, l, L_u) (This means L_u either reaches a non-connector or a used connector, then we will use the subroutine Sampling-AH to deal with it.)
    end if
end while
return the endpoint of L_u

Algorithm 7 Sampling-AH(x, T, S, l, L_u)

while $|L_u| < l$ **do**
    Subphase (2.1) $E_u \leftarrow$ sample s edges (with repetition) for every node $u \in S$. (using two rounds)
    Extend L_u using edges sampled in Subphase (2.1). (When visiting a node $v \in S$ for the k-th time, use the k-th sampled edge from E_v.)
    $x \leftarrow$ new endpoint after extending L_u (Then there are three cases.)
    (1) if $(x \in S)$, **continue** (L_u is still in S, at least s progress is made.)
    (2) if ($x \in T$ and $x \notin S$), **return** (This means L_u reaches an unused connector and we can use it to extend the walk.)
    (3) if ($x \notin T$ and $x \notin S$), $S \leftarrow S \cup \{x\}$, **continue** ($L_u$ reaches a new non-connector.)
end while
return the endpoint of L_u

4.1.1 Detailed Implementation

Subphase (1.1) in the Sampling-AS Algorithm

We assume all nodes are pre-labeled with distinct labels (usually a sequence of integers from 1 to n). We also randomly pick a hash function that uniformly maps the labels of nodes to some buckets.

For each sampled edge, we use the labels of both ends as the input for the hash function. During the sampling process, no matter how many times a node's label is used as the input in the hash function, the label will always be mapped to the same bucket.

Before the sampling starts, we randomly choose α fraction of buckets. During the sampling process, if a node's label is in one of the buckets chosen, then we store the node into T. Otherwise, we do not store the node. By doing so, each node appearing within the round has probability α to be sampled into T. Moreover, only nodes that are sampled are stored.

By Lemma 5, with probability at least $1-n^{-1}l^{-1}$, we will see all edges in E in Subphase (1.1). Thus, w.h.p., each node is considered and sampled with probability α because the graph is assumed to be connected. We use Q_0 to represent the event that some node does not show up in Subphase (1.1) and $Pr[Q_0] \leq n^{-1}l^{-1}$.

The Technique to Sample Sufficient Edge Samples

Before we describe other subphases, we want to explain the technique that is used to sample a sufficient number of edges for vertices of low degree. This technique is called *Sufficient-Sampling* and it is used in Subphases (1.2) and (2.1). Since the technique is not too complicated, it might have been used by other researchers.

For any node v, we assume that m_v samples are required. Sufficient-Sampling uses a counter f_v to count the number of edge samples still required for v during the procedure. Sufficient-Sampling also uses a multi-set E_v to store sampled edges for node v (The k-th item in E_v means the k-th item that is added into E_v). Initially, $f_v = m_v$ and E_v is empty. By the end of Sufficient-Sampling, $f_v = 0$ and E_v contains m_v edge samples from v.

When $G(V, E)$ is a directed graph, if each sampled edge e's source is some vertex $v \in U$, we store e into E_v and deduct 1 from f_v. If f_v becomes 0, we no longer consider new edge samples for v (v already has m_v edge samples).

For each sampled edge e in undirected graphs, we randomly and uniformly select one end of e as the source. Then we follow the same steps as for directed graphs, which ensures that one edge sample can be used to extend only one random walk.

The procedure so far is named *Normal-sampling*.

After Normal-sampling, for any vertex v for which $f_v > 0$ (we still need f_v edge samples from v and we sample them from edges we have stored for v in Normal-sampling), we uniformly sample f_v edge samples (with repetition) from the distinct edges in E_v and add

the sampled edges to E_v. This step is called *Re-sampling*. Now, each vertex $v \in U$ has m_v edge samples in E_v.

We let R^* denote the event that some edge in E is not output by the oracle in Normal-sampling. Sufficient Sampling works unless the "bad" event R^* happens. The core result of this technique is given in Lemma 6.

Lemma 6. *For a graph $G(V, E)$ in the edge sample model, given a vertex set $U \subseteq V$, each vertex $v \in U$ has an associated value m_v denoting the number of edge samples required for v. For each node $v \in U$, Sufficient-Sampling independently and uniformly samples m_v edges from all outgoing edges of v, unless the event R^* happens. The probability that R^* happens is at most $n^{-2}l^{-1}$. In addition, Sufficient-Sampling always uses two rounds and $O(\sum_{v\in U}(1 + m_v))$ space,*

Proof. We first prove that if R^* does not happen, the edge samples in each E_v are independent and uniform samples from all outgoing edges of v.

Considering a node v, there are two cases where E_v gets an edge sample e.

Case 1. e is sampled during the Normal-sampling step.

Case 2. e is uniformly selected from the distinct edges of E_v in the Re-sampling step.

In Case 1, since any edge e only has one source, say v, e is used as an outgoing edge sample for only one vertex v. Therefore, edge samples in Case 1 are never reused. In addition, because the oracle outputs each outgoing edge of v with the same probability, e is a uniform sample from all outgoing edges of v.

In Case 2, given that R^* does not happen, all edges of E are sampled during Normal-sampling. Consequently, for any vertex v, E_v contains all outgoing edges of v. Hence, sampling e uniformly from the distinct edges of E_v is equivalent to sampling e uniformly from all outgoing edges of v. Moreover, e is used by only one vertex v.

In both cases, an edge sample e is never reused. Therefore, all edge samples are independent.

Secondly, we prove the space complexity in Lemma 6. For each vertex v, the counter f_v costs $O(1)$ space and storing sampled edges also costs $O(m_v)$ space. Hence, the total space complexity is $O(\sum_{v\in U}(1 + m_v))$.

Finally, by Lemma 5, the probability that R^* happens is at most $n^{-2}l^{-1}$. □

Subphase (1.2) in the Sampling-AS Algorithm

We divide Subphase (1.2) into a sequence of w iterations. In each iteration, we extend all short random walks by one step. After w iterations, all short random walks have length w.

Within a specific iteration, we denote the current distinct endpoints of all short random walks by U_1. For each vertex $v \in U_1$, the number of random walks at v is denoted by $m_v{}'$

(we need $m_v{}'$ edge samples for v to extend all random walks currently stopping at v). Then we use Sufficient-Sampling with $U = U_1$ and $m_v = m_v{}'$ for each $v \in U$. By Lemma 6, if R^* does not happen, each vertex $v \in U_1$ gets sufficient edge samples to extend $m_v{}'$ random walks stopping at v using two rounds.

We let R_1 denote the event that in some iteration, the "bad" event R^* of Sufficient Sampling happens. Now, we present the analysis for Subphase (1.2) in Lemma 7.

Lemma 7. *Given any undirected connected graph G with n vertices, Subphase (1.2) of Sampling-AS samples the endpoints of $|T|$ independent w-length random walks, unless the event R_1 happens. Event R_1 happens with probability at most $\frac{1}{n}$. In addition, Subphase (1.2) always uses $2w$ rounds and $O(|T|)$ space,*

Proof. In w iterations, we use Sufficient-Sampling w times. By Lemma 6 and the union bound, $Pr[R_1] \leq w \cdot n^{-2} l^{-1} \leq \frac{1}{n}$.

If R_1 does not happen, the future state of any walk depends only on the current state because edge samples are independent and are never reused. Therefore, for any w-length walk, the stochastic process that generates the walk satisfies Condition RW1.

Also, if R_1 does not happen, all edge samples of a vertex v are uniform samples from all outgoing edges of v. When any w-length walk visits a vertex v, it chooses each outgoing edge of v with the same probability to move forward. As a result, for any w-length walk, the stochastic process that generates the walk also satisfies Condition RW2.

In conclusion, if R_1 does not happen, the generated w-length walks are random walks.

Finally, we analyze the space complexity for Subphase (1.2). In each iteration, the total number of short random walks is $|T|$, which makes $\sum_{v \in U_1} m_v{}' = |T|$. Hence, the space used in one iteration is $O(|T|)$ by Lemma 6. Because each iteration reuses the space, the total space complexity is $O(|T|)$.

□

Subphase (2.1) in the Sampling-AS Algorithm

In Subphase (2.1), we use Sufficient-Sampling by setting $U = S$ and $m_v = s$ for each vertex $v \in U$.

We give the analysis for Subphase (2.1) in Lemma 8.

Lemma 8. *Given any undirected graph G with n vertices and any $l \geq 0$, Subphase (2.1) of Sampling-AS independently and uniformly samples s edges from all outgoing edges for each node in S, unless the event R^* happens. The event R^* happens with probability at most $\frac{1}{nl}$. In addition, Subphase (2.1) always uses two rounds and $O(|S| \cdot s)$ space,*

The proof for Lemma 8 is omitted since it follows directly from Lemma 6.

4.1.2 Analysis

We let R_2 denote the event that in some invocation of Subphase (2.1) event R^* happens. R will be the union of the "bad" events R_1 and R_2. The following theorem describes the core results of our analysis. Its proof is given in Sections (a) and (b).

Theorem 3. *Given any undirected connected graph G with n vertices and any integer $l \geq 0$, the Sampling-AS algorithm always samples the endpoint of a l-length random walk, unless the event R happens (R happens with probability at most $\frac{2}{n}$). The algorithm uses $\tilde{O}\left(\alpha n + \frac{l^{\frac{1}{2}}}{\alpha}\right)$ space and terminates in $\tilde{O}\left(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha}\right)$ rounds w.h.p. In particular, if $l = O(n)$ and $\alpha = n^{-\frac{1}{4}}$, then the space complexity is $\tilde{O}(n^{\frac{3}{4}})$ and the number of rounds is $\tilde{O}(n^{\frac{3}{4}})$ w.h.p.*

(a) Correctness and Error Bounds

After each invocation of Subphase (2.1), the Sampling-AS algorithm extends L_u by at least one step. Therefore, the total number of invocations of Subphase (2.1) is bounded by l. By the union bound and Lemma 8, $Pr[R_2] \leq l \cdot \frac{1}{nl} = \frac{1}{n}$.

By the union bound, Lemma 7 and the analysis above,

$$Pr[R] \leq Pr[R_1] + Pr[R_2] \leq \frac{2}{n}. \tag{4.4}$$

This completes the proof of the error bounds for the Sampling-AS algorithm.

L_u is constructed by two types of short walks.

Type 1 w-length walks prepared in Subphase (1.2)

Type 2 walks prepared in each invocation of Sampling-AH

If event R does not happen, Lemma 7 implies that walks of Type 1 are independent random walks. Now we prove that walks of Type 2 are also random walks. We look at some walk L^* of Type 2 which is generated in one invocation of Sampling-AH. Because L^* uses the k-th edge sample of v if L^* visits v for the k-th time, no edge sample is reused. Hence, at each step, the future state of L^* only depends on the current state, and the stochastic process $\widetilde{L^*}$ that generates L^* satisfies Condition RW1. If the event R does not happen, when L^* visits a vertex v, it chooses each outgoing edge of u with the same probability. As a result, $\widetilde{L^*}$ also satisfies Condition RW2. In conclusion, L^* is a random walk if R does not happen.

Moreover, as each edge sample is used by at most one walk and a walk is used at most once, all walks of Type 1 and Type 2 are independent. By Lemma 1, the walk L_u, generated by stitching these independent short random walks is also a random walk (given the condition that the event R does not happen).

(b) Probabilistic Bounds on Space and the Number of Rounds

First of all, we prove the following lemma.

Lemma 9. *Given the condition that the event Q_0 does not happen during the execution of the Sampling-AS algorithm, if L_u stops at a node for the first time, with probability α, the node is a connector.*

Proof. If the event Q_0 does not happen, each node is sampled as a connector with probability α in Subphase (1.1). Under such condition, the scenario in the Sampling-AS algorithm is the same as that in the Streaming-S algorithm. Therefore, we omit the rest of the proof since it is the same as the proof for Observation 1. □

Using Lemma 9, Lemma 10 can be proved.

Lemma 10. *For the sequence of nodes added to S, there are at most $\tilde{O}(\frac{1}{\alpha})$ consecutive non-connectors before a connector occurs w.h.p.*

Proof. We also omit most of the proof for Lemma 10 as it is almost the same as Lemma 4 conditioned on $\overline{Q_0} \cap \overline{Q_2}$. The probability that event Q_0 or event Q_2 happens can be easily bounded by $\frac{2}{n}$ using the union bound. □

We use Q' to represent the union of events Q_0, Q_1 (the event that the number of connectors is not $\tilde{O}(\alpha n)$) and Q_2 (the event that there is some connector $v \in S$ for which there are more than $\tilde{O}(\frac{1}{\alpha})$ non-connectors added before v). By the union bound, $Pr[Q'] \leq \frac{3}{n}$.

The maximal number of connectors used is $\frac{l}{w}$ by Lemma 2. By Lemma 10, $|S|$ is at most $\tilde{O}(\frac{l}{\alpha w})$ if the event Q' does not happen. Since each node in S stores s edge samples, the total space required for the subroutine Sampling-AH is $\tilde{O}(\frac{ls}{\alpha w})$ w.h.p.

Therefore, the total space needed for the Sampling-AS algorithm is $\tilde{O}(\alpha n + \frac{ls}{\alpha w})$ w.h.p. If $l = O(n), \alpha = n^{-\frac{1}{4}}, s = n^{\frac{1}{4}}, w = n^{\frac{3}{4}}$, the space complexity is $\tilde{O}(n^{\frac{3}{4}})$ w.h.p., which is sublinear.

Next, the number of rounds required can be bounded. First of all, sampling connectors uses one round and generating w-length walks for connectors takes $2w$ rounds. Using analysis similar to that in the Streaming-S algorithm, the total number of rounds for Phase 2 is $\tilde{O}(\frac{l}{\alpha w} + \frac{l}{s})$ if the event Q' does not happen.

From the analysis above, the total number of rounds required by the Sampling-AS algorithm is $\tilde{O}(w + \frac{l}{\alpha w} + \frac{l}{s})$ w.h.p. If $l = O(n), \alpha = n^{-\frac{1}{4}}, s = n^{\frac{1}{4}}, w = n^{\frac{3}{4}}$, the total number of rounds required is $\tilde{O}(n^{\frac{3}{4}})$ w.h.p., which is also sublinear.

Based on the previous analysis for space complexity and the number of rounds, letting $w = l^{\frac{3}{4}}$ and $s = l^{\frac{1}{4}}$, we obtain Theorem 3.

Space Improvement

We name the modified Sampling-AS algorithm Sampling-SAS (short for Sampling Space Saving Atlantic Single Random Walk) and the modified subroutine Sampling-SAH. Sampling-SAS uses an idea similar to the one in the Streaming-SS algorithm by changing Subphase (2.1) to save space.

To identify vertices of low degree, the Streaming-SS algorithm uses an additional pass to count the out-degree of each vertex in S. However, this technique does not work in the edge sampling model. First of all, the oracle might output repeated edges in the edge sampling model. To count distinct outgoing edges of a vertex, each edge will need to be stored, which costs much space. Moreover, even after a large amount of queries, we cannot make sure if all outgoing edges of a vertex have been sampled. Therefore, even if we store all edges of a vertex u output by the oracle, the out-degree of u may still be underestimated.

Sampling-SAS skips the Re-sampling step of Sufficient-Sampling in Subphase (2.1). After the execution of the modified Sufficient-Sampling, vertices in S have at most s edges. Vertices in S can be further divided into two subsets F and M. For a vertex u, if the number of sampled edges for u (which is $|E_u|$) is less than s, u is put into F. Otherwise, u is put into M.

Later on when L_u visits some node $v \in S$ for the k-th time, if $v \in F$, L_u is extended using an uniform edge sample from distinct outgoing edges of E_v. If $v \in M$, L_u is extended using the k-th edge sample of E_v. The details for the subroutine Sampling-SAH are presented in the pseudo code below.

Algorithm 8 Sampling-SAH(x, T, S, L_u, l)

procedure SAMPING-SAH
  while $|L_u| < l$ **do**
    Subphase (2.1) Sample at most s edge samples for each node in S, and divide all the nodes in S into two subsets F and M. F contains nodes whose number of edge samples is less than s, while M contains nodes with s edge samples. (using two rounds)
    Extend L_u by random walking using previously stored edges. (When visiting a node $v \in F$, use the edge sampled uniformly from all distinct sampled edges of v; when visiting a node v' in M for the k-th time, use the k-th sampled edge from v'.)
    $x \leftarrow$ new endpoint after extending L_u (Then there are three cases.)
    (1) if ($x \in S$), **continue** { no new node is seen, at least s progress is made.}
    (2) if ($x \in T$ and $x \notin S$), **return** {this means that x is a node that has not been seen in the walk so far, and x is among the set of nodes sampled initially; therefore, the w-length walk from x has not been used}
    (3) if ($x \notin T$ and $x \notin S$), $S \leftarrow S \cup \{x\}$, **continue** {this means that x is a new node that has not been visited in this invocation, and x is not in the initial set sampled nodes T }
  end while
  return final destination of L_u
end procedure

In the Streaming-SS algorithm, we can know whether a vertex v has out-degree less than s or not by counting the out-degree for each vertex using an additional pass. If the out-degree of v is less than s, the Streaming-SS algorithm only stores distinct edges for v rather than storing s edge samples with repetition for v. For each vertex $v \in S$ in the edge sampling model, the Sampling-SAS algorithm needs to store all edge samples with repetition for u in Subphase (2.1), because we need repeated edge samples if u has high out-degree. Compared to the Streaming-SS algorithm, the Sampling-SAS algorithm avoids counting the out-degree for each vertex and thus reduces the number of rounds required. However, the Sampling-SAS algorithm stores repeated edge samples for vertices of low out-degree and this increases space complexity. Next we show that compared to the Streaming-SS algorithm, the Sampling-SAS algorithm only increases the space required slightly.

Lemma 11. *For any edge e in a graph, the number of times that e is sampled within two rounds is $6\log(mnl)$ w.h.p.*

Proof. First of all, we consider a specific edge e. We let X_i be 1 if the i-th query is e and 0 otherwise. We let $X = \sum_{i=1}^{2m\log(mnl)} X_i$. Clearly, X is the number of times that e is sampled within two rounds and $E[X] = \frac{1}{m} \cdot 2m\log(mnl) = 2\log(mnl)$. Since all the X_i's are independent, identical and 0-1 random variables, we can apply Chernoff bound for X and get

$$P[X > (1+\delta)E[X]] \leq e^{-\frac{\delta^2}{2+\delta}2\log(mnl)}. \tag{4.5}$$

Set $\delta = 2$, then

$$P[X > 3E[X] = 6\log(mnl)] \leq e^{-2\log(mnl)} = (mnl)^{-2}. \tag{4.6}$$

Let Q_3^* denote the event where at least one edge in E is sampled more than $6\log(mnl)$ times within two rounds. By the union bound,

$$P[Q_3^*] \leq \sum_{i=1}^{|E|}(mnl)^{-2} \leq m \cdot (mnl)^{-2} \leq (nl)^{-1}. \tag{4.7}$$

□

Let Q_3 be the event that during some Subphase (2.1) in the entire Sampling-SAS algorithm, some edge is sampled more than $6\log(mnl)$ times. By the union bound, $Pr[Q_3] \leq l \cdot \frac{1}{nl} = \frac{1}{n}$ because the number of times that Subphase (2.1) is executed is at most l. In addition, denote the union of events Q' and Q_3 as Q_s, $Pr[Q_s] \leq \frac{4}{n}$ (also by the union bound).

Therefore, if event Q_s does not happen, the Sampling-SAS algorithm only increases the space storing edges of low-degree vertices by a factor of $6\log(mnl)$. Moreover, as

the $\tilde{O}$ symbol hides log terms, space complexities are the same between the Streaming-SS algorithm and the Sampling-SAS algorithm.

Compared to the Sampling-AS algorithm, the number of rounds required for the Sampling-SAS algorithm does not increase. The space required is at most $|S|s$ in Subphase (2.1). The space improvement effect of the Sampling-SAS algorithm also depends highly on the graph structure, especially on the degree distribution. This will be discussed in detail in Chapter 5.

4.2 The Las Vegas Algorithm

The Sampling-AS algorithm samples the endpoint of a random walk of length l if event R does not happen. Although R seems unlikely, the correctness of the algorithm cannot be guaranteed since it is unknown if R happens or not. Meanwhile, in some scenarios, we do need an algorithm that always samples the endpoint of a random walk of length l. Therefore, a Las Vegas algorithm called Sampling-LS (short for Sampling Las Vegas Single Random Walk) is introduced which samples the endpoint of a l-length random walk. The Sampling-LS algorithm's output is always correct, while the number of rounds required and space complexity are random variables. Although the Sampling-LS algorithm can also be applied to directed graphs, its probabilistic bounds on the number of rounds and space required is only proven on undirected graphs because the technique we use for undirected graphs cannot be applied to directed graphs. This will be further explained at the end of Section 4.2.1.

The Sampling-LS algorithm modifies Subphases (1.2) and (2.1). When many edge samples are required by the vertices of low-degree in both subphases, the Sampling-LS algorithm no longer uses Sufficient-Sampling that might cause event R. The algorithm keeps querying the oracle until each vertex of low-degree gets sufficient amount of edge samples required. Obviously, the algorithm needs more queries than the Sampling-AS algorithm. Nevertheless, we prove that in undirected graphs, the number of queries needed will not increase significantly. Below is the detailed pseudo code of the Sampling-LS algorithm. We will further explain Subphase (1.2) and Subphase (2.1) in Section 4.2.1.

4.2.1 Detailed Implementation

In this section, the key subphases in the Sampling-LS algorithm will be explained and the number of rounds and space required will be bounded.

Algorithm 9 Sampling-LS(u, l)

Input: Starting node u and desired walk length l.
Output: The endpoint of L_u (the random walk from u of length l)
Phase 1 starts
Subphase (1.1) Sample each distinct node appearing in a round independently with probability α, and store the sampled nodes into a set T.
Subphase (1.2) Perform walks of length w from every node in T. Let $W[t]$ be the endpoint of the walk of length w starting from $t \in T$. (completes in $6w \log(nl)$ rounds w.h.p.)
Phase 2 starts
Initialize L_u to be a zero length walk starting at u and let $x \leftarrow u$
$S \leftarrow \{\ \}$ (which is used to store vertices L_u has stopped at)
while $|L_u| < l$ **do**
    if $x \in T$ and $x \notin S$ **then**
        Extend L_u by appending the available w-length walk (implicitly by setting the current endpoint x of L_u to be the endpoint of the w-length walk starting from x). $S \leftarrow S \cup \{x\}$, and let $x \leftarrow W[x]$.
    else
        Sampling-LH(x, T, S, l, L_u) (This means L_u either reaches a non-connector or a used connector, then we will use the subroutine Sampling-LH to deal with it.)
    end if
end while
return the endpoint of L_u

Algorithm 10 Sampling-LH(x, T, S, l, L_u)

while $|L_u| < l$ **do**
    Subphase (2.1) Sample at most s edges into E_v (with repetition) for each node $v \in S$ using $64\sqrt{s+1}\log(nl)$ rounds.
    Extend L_u by using edges sampled in Subphase (2.1). (When visiting a node $v \in S$ for the k-th time, use the k-th sampled edge from E_v.)
    $x \leftarrow$ new endpoint after extending L_u (Then there are three cases.)
    (1) if ($x \in S$), **continue** (L_u is trapped in S, at least s progress is made w.h.p.)
    (2) if ($x \in T$ and $x \notin S$), **return** (This means L_u reaches an unused connector and we can use it to extend.)
    (3) if ($x \notin T$ and $x \notin S$), $S \leftarrow S \cup \{x\}$, **continue** ($L_u$ reaches a new non-connector.)
end while
return the endpoint of L_u

Subphase (1.2) in the Sampling-LS Algorithm

For each short random walk, a counter is used to store the current length. Subphase (1.2) is still divided into a sequence of w iterations. Within each iteration, a short random walk gets exactly one edge sample and thus moves one step forward.

Within the i-th iteration, initially all short random walks have length $i-1$. If a new edge sample e cannot be used to extend any $(i-1)$-length random walk, then no actions will be performed. If e can be used, we uniformly select a $(i-1)$-length random walk L' from all $(i-1)$-length random walks that can use e. Then L' is extended using e and the counter of L' is set to i. We keep querying the oracle for new edge samples until each random walk has length i.

If the number of rounds required for a single iteration is bounded, the total number of rounds required for Subphase (1.2) can also be bounded. However, when trying to bound the number of rounds required for one iteration, the following challenge is encountered.

Assuming that Subphase (1.2) needs to perform k short random walks, it is possible that multiple random walks are all at some node v of low-degree at the beginning of some iteration. Thus, within the iteration, many queries are required to get sufficient amount of edge samples from v since sampling one edge from v has low probability. In the worst case, the k random walks are all at some node v' of degree 1. Using k rounds, each random walk is only extended by length 1. Therefore, in the worst case, performing the k random walk processes in Subphase (1.2) requires $O(kw)$ rounds, which is no better than the trivial algorithm that performs k random walk processes one after another.

Although the worst case can happen, the next lemma proves that the worst case is unlikely.

Lemma 12. *Given an undirected graph G with n vertices, any non-negative integer $k \leq n$ and any integer $w \geq 0$, Subphase (1.2) of the Sampling-LS algorithm always samples the endpoints of k independent random walks of length w. In addition, Subphase (1.2) terminates in $6w\log(nl)$ rounds and uses $O(k)$ space w.h.p.*

Proof. First of all, we prove that k w-length walks generated are random walks. We look at one walk L^*. Since an edge sample is never reused, at each step, the future state of L^* only depends on the current state. Therefore, the stochastic process $\widetilde{L^*}$ that generates L^* satisfies Condition RW1. Meanwhile, when L^* visits a vertex u, it chooses each outgoing edge of u with the same probability. As a result, $\widetilde{L^*}$ also satisfies Condition RW2. In conclusion, L^* is a random walk and all generated w-length walks are random walks.

Next, we bound the number of rounds required using the idea in the proof of Lemma 3.2 in paper [15].

In Subphase (1.2), one random walk is performed for each connector. For better analysis, assume there is another algorithm called *Sampling-I* and we couple Subphase (1.2) of Sampling-LS and Sampling-I. Sampling-I not only performs all random walks required by

Subphase (1.2) of Sampling-LS, but also performs additional random walks. For each node $v \in V$, Sampling-I performs $d(v)$ random walks from v. The random walks performed in the Sampling-I algorithm can be partitioned into two sets. The first set contains random walks required for connectors in Subphase (1.2) of Sampling-LS. The second one contains additional random walks added so that each node v has $d(v)$ random walks from v. In one iteration, the number of rounds required by Sampling-I is no less than that required by Sampling-LS, since Sampling-I needs additional edge samples for extra random walks.

If there is an upper bound on the number of rounds required for one iteration by Sampling-I, this bound is also an upper bound on the number of rounds required for one iteration in Subphase (1.2). Then, we focus on the Sampling-I algorithm and bound the number of rounds required in one iteration.

We define $X_i^j(e)$ to be a random variable whose value is 1 if the i-th random walk passes through edge e in the j-th iteration, otherwise its value is 0. Then we let $X^j(e) = \sum_i X_i^j(e)$. Since all $X_i^j(e)$s are independent, Chernoff bound can be applied to upper bound $X^j(e)$, which can prove that w.h.p. every edge does not need to be sampled many times to extend all random walks.

Claim 2. *For any edge e and any j, $E[X^j(e)] = 2$.*

Proof (of Claim 2). Since each node v has $d(v)$ random walks in the beginning, the number of random walks starting at v is proportional to its stationary distribution $\frac{d(v)}{2m}$. Therefore, after j iterations, the expected number of random walks that are at v is still $d(v)$.

For an edge $e = \{x, y\}$ in the j-th iteration, the number of random walks passing through e comes from two ends x and y. The expected number of random walks passing from x to y is $d(x) \cdot \frac{1}{d(x)} = 1$. After considering the random walks from y to x, $E[X^j(e)] = 2$. □

By Chernoff bound from Theorem 4.4 of [37], for any edge e and any iteration j (since n is large, $R = 3\log(nl) > 6E[X^j(e)]$),

$$Pr[X^j(e) \geq 3\log(nl)] \leq 2^{-R} = (nl)^{-3}. \tag{4.8}$$

We let Q_4^* denote the event that within one iteration, there exists an edge e and an integer $1 \leq j \leq w$ such that $X^j(e) \geq 3\log(nl)$. By the union bound, $Pr[Q_4^*] \leq m \cdot w \cdot (nl)^{-3} \leq n^2 \cdot l \cdot (nl)^{-3} \leq \frac{1}{nl}$. Then we use $Q_4{}'$ to represent the event that in some iteration of w iterations, Q_4^* happens. Also by the union bound, $Pr[Q_4{}'] \leq w \cdot \frac{1}{nl} \leq \frac{1}{n}$. Therefore, if $Q_4{}'$ does not happen, we need at most $3\log(nl)$ edge samples for each edge e to extend all random walks in one iteration in the Sampling-I algorithm.

By Lemma 5, using two rounds, we get at least 1 edge sample for each edge with probability no less than $1 - (nl)^{-2}$. In order to get $3\log(nl)$ edge samples for each edge, we use $6\log(nl)$ rounds in each iteration. If we denote the event that within w iterations, some edge is not sampled for $3\log(nl)$ times in some iteration by $Q_4{}''$. By the union bound,

$Pr[Q_4''] \leq w \cdot 6\log(nl) \cdot (nl)^{-2} \leq \frac{1}{n}$. Denoting the union of event Q_4' and Q_4'' by Q_4, each iteration requires $6\log(nl)$ rounds if event Q_4 does not happen. Moreover, $Pr[Q_4] \leq \frac{2}{n}$.

In conclusion, Subphase (1.2) requires $6w\log(nl)$ (which is $\tilde{O}(w)$) rounds w.h.p.

Finally, we prove the space complexity is $O(k)$. Within each iteration, we use $O(k)$ space to store the current node and the current length for each random walk. Therefore, the total space required is $O(k)$. □

Note that the previous proof only works in undirected graphs. In directed graphs, it is possible that the in-degree of a node v is much more than the out-degree of v. If there is a random walk visiting v from each incoming edge in the last iteration, then the current iteration requires a large number of queries to sample enough edges to extend all random walks at v. A counterexample can be easily found to prove that Lemma 12 is not valid in directed graphs. Consider a directed graph like a star, there is one vertex a as a hub with only one outgoing edge pointing to another vertex b. All other vertices (including b) contain one outgoing edge pointing to a. If a random walk is started from each vertex, these random walks will visit a and approximately n rounds are required to extend all random walks by length 1.

Subphase (2.1) in the Sampling-LS Algorithm

In Subphase (2.1), we do not prepare s edge samples for each node but use the following way to sample at most s edges for nodes in S. We query the oracle for $64\sqrt{s+1}\log n$ rounds. For each vertex $v \in S$, a multi-set E_v can be used to store sampled edges of v (with repetition), and it stops storing new sampled edges of v if the size of E_v equals s. After the process completes, each vertex in S has at most s edge samples. Moreover, vertices of low-degree are likely to have less than s edge samples.

Lemma 3.17 in [15] is restated since it is the starting point of our analysis. We let $N_t^x(y)$ denote the number of times that a random walk visits vertex y after t steps, given the random walk starts at vertex x.

Lemma 13 (Lemma 3.17 in [15])**.** *For $t = O(m^2)$ and any vertex $y \in V$, the random walk that starts at $x \in V$ satisfies*

$$Pr[N_t^x(y) \geq 32d(y)\sqrt{t+1}\log n] \leq \frac{1}{n^2}. \tag{4.9}$$

By Lemma 13, a random walk of length s is unlikely to visit a vertex of low degree s times. Then, Lemma 14 can be proven, which shows that for any vertex $v \in S$, the amount of edge samples stored in E_v after $64\sqrt{s+1}\log n$ rounds is sufficient to extend a random walk by length s w.h.p.

Lemma 14. *During $64\sqrt{s+1}\log(nl)$ rounds, Subphase (2.1) of Sampling-LS uses at most $O(|S|\cdot s)$ space to store edge samples. The stored edge samples are sufficient to extend a random walk by length s w.h.p.*

Proof. According to Lemma 13, by replacing the parameter n with nl in Inequality (4.9), for a random walk of length s, a vertex y of degree $d(y)$ will be visited at most $32d(y)\sqrt{s+1}\log(nl)$ times, otherwise we say event Q_5^* happens (Q_5^* happens with probability at most $(nl)^{-2}$). For each vertex y, w.h.p. $deg(y)$ edges are sampled for y using two rounds. Therefore, we use $64\sqrt{s+1}\log(nl)$ rounds to get the sufficient amount of edge samples for each vertex y. During Subphase (2.1), every edge in E is output by the oracle in every 2 rounds w.h.p, otherwise we denote the event by $Q_5{}''$.

Let $Q_5{}'$ denote the event that Q_5^* happens in some invocation of Sampling-LH, by the union bound $Pr[Q_5{}'] \leq l \cdot \frac{1}{(nl)^2} \leq \frac{1}{n}$. Following a similar analysis to that of Subphase (1.2), we get $Pr[Q_5{}''] \leq l \cdot (nl)^{-2} \leq \frac{1}{n}$. Therefore, throughout the Sampling-LS algorthm, Subphase (2.1) uses $\tilde{O}(\sqrt{s})$ rounds, unless event $Q_5{}''$ happens. More importantly, this amount of edge samples is sufficient to perform a random walk of length s, if event $Q_5{}'$ does not happen. Denote the union of $Q_5{}'$ and $Q_5{}''$ by Q_5 and $Pr[Q_5] \leq \frac{2}{n}$ by the union bound.

Then, we prove that space complexity is $O(|S|\cdot s)$. Before starting sampling, $O(|S|)$ space is used to store all nodes in S. For a vertex y of high degree, $32d(y)\sqrt{s+1}\log(nl)$ may be greater than s, but we only store at most s edges for y. Therefore, $\min\{s,\ 32d(y)\sqrt{s+1}\log(nl)\}$ edges are required for each vertex $y \in S$. As a result, the total space required is still bounded by $O(|S|\cdot s)$.

□

4.2.2 Analysis

The following theorem that describes the core result of our analysis will be proven in Sections (a) and (b).

Theorem 4. *Given an undirected graph G with n vertices and any $l \geq 0$, the Sampling-LS algorithm always samples the endpoint of a single random walk with length l. The algorithm uses $\tilde{O}(\alpha n + \alpha l)$ space and terminates in $\tilde{O}\left(l^{\frac{4}{5}} + \frac{l^{\frac{3}{5}}}{\alpha}\right)$ rounds w.h.p. In particular, if $l = O(n)$ and $\alpha = n^{-\frac{1}{5}}$, space complexity is $\tilde{O}(n^{\frac{4}{5}})$ and the number of rounds required is $\tilde{O}(n^{\frac{4}{5}})$ w.h.p.*

(a) Correctness

We follow the argument in Subsection (a) of Section 4.1.2.

Lemma 12 proves that walks of Type 1 (see page 33) are independent random walks. Now we prove that walks of Type 2 (also see page 33) are also random walks. Consider a walk L^* of Type 2 which is generated in one invocation of Sampling-LH. No edge sample is

reused, since L^* uses the k-th edge sample of u if L^* visits u for the k-th time. Hence, at each step, the future state of L^* only depends on the current state. Therefore, the stochastic process $\widetilde{L^*}$ that generates L^* satisfies Condition RW1. Meanwhile, when L^* visits a vertex u, L^* chooses each outgoing edge of u with the same probability. As a result, $\widetilde{L^*}$ also satisfies Condition RW2. In conclusion, L^* is a random walk if R does not happen.

Moreover, as each edge sample is used by at most one short random walk and each short random walk is used at most once, all random walks of Type 1 and Type 2 are independent. By Lemma 1, the walk L_u, generated by stitching these independent short random walks is also a random walk.

In conclusion, L_u generated by Sampling-LS is always a random walk.

(b) Probabilistic Bounds on Space and Rounds

We use the same event Q' (see page 34) in the Sampling-AS algorithm. As analyzed before, Q' is the union of events Q_0, Q_1 and Q_3, and $Pr[Q'] \leq \frac{3}{n}$. W.h.p., Q' does not happen.

Space complexity will be analyzed first. By Lemma 3, we need $\tilde{O}(\alpha n)$ space to store endpoints of w-length random walks w.h.p. The maximal number of connectors used is $O(\frac{l}{w})$ by Lemma 2. According to Lemmas 10 and 14, the space required in Subphase (2.1) is $\tilde{O}(\frac{l}{\alpha w}s)$ w.h.p. Therefore, the total space required is $\tilde{O}(\alpha n + \frac{l}{\alpha w}s)$ w.h.p.

Next, we bound the number of rounds required. Subphase (1.1) uses one round. If event Q_4 does not happen, Subphase (1.2) uses $6w\log(nl)$ rounds by Lemma 12. The number of times that Subphase (2.1) executes is bounded by $\tilde{O}(\frac{l}{s} + \frac{l}{\alpha w})$ using the same analysis as that of the Sampling-AS algorithm, unless event Q' happens. If Q_5 does not happen, Subphase (2.1) requires $\tilde{O}(\sqrt{s})$ rounds by Lemma 14. We denote the union of events Q', Q_4 and Q_5 by Q_L. By the union bound, $Pr[Q_L] \leq \frac{3}{n} + \frac{2}{n} + \frac{2}{n} = \frac{7}{n}$. Therefore, the total number of rounds required is $\tilde{O}(w + (\frac{l}{s} + \frac{l}{\alpha w})\sqrt{s})$ w.h.p.

If we set $w = l^{\frac{4}{5}}$ and $s = \alpha^2 l^{\frac{4}{5}}$, the total space required is $\tilde{O}\left(\alpha n + \alpha l\right)$ and the total number of rounds required is $\tilde{O}\left(l^{\frac{4}{5}} + \frac{l^{\frac{3}{5}}}{\alpha}\right)$ w.h.p. When $l = O(n)$ and $\alpha = n^{-\frac{1}{5}}$, space complexity is $\tilde{O}(n^{\frac{4}{5}})$ and the total number of rounds is $\tilde{O}(n^{\frac{4}{5}})$. Compared to the Sampling-AS algorithm, space complexity and the total number of rounds required for the Sampling-LS algorithm increase by $\tilde{O}(n^{\frac{1}{20}})$.

This completes the proof of Theorem 4.

Chapter 5

Special Classes of Graphs

In Chapter 3 and Chapter 4, we described several algorithms that can sample the endpoint of a random walk in the graph streaming model and the edge sampling model. In addition, we gave probabilistic bounds on the space and the number of passes/rounds required. We also provided a technique to save space for these algorithms. In this chapter, we analyze the performance of these algorithms on special classes of graphs such as regular graphs, random graphs and fast mixing graphs.

5.1 Regular Graph

A regular graph is a graph each vertex of which has the same degree. For a directed regular graph, we additionally require that for each vertex, its in-degree equals to its out-degree. A regular graph whose vertices have degree k is called a k-regular graph.

Throughout our analysis, we assume k is a constant and it is known. In this case, only $O(1)$ unit of space is required to store all the edges incident on a vertex. In other words, instead of using s space in Subphase (2.1) for each vertex in S, only $O(1)$ space is needed by using the space-saving technique stated in earlier chapters. Moreover, because the change of s's value has no impact on the amount of space required, an increase in s would also reduce the number of passes/rounds required.

The Streaming-SS algorithm

First of all, we consider the algorithm Streaming-SS that uses our space saving technique in the graph streaming model. The following lemma states the performance of Streaming-SS.

Lemma 15. *Given a k-regular connected graph with n vertices, the Streaming-SS algorithm always samples the endpoint of a random walk with length l. W.h.p., it uses $\tilde{O}(\alpha n + \sqrt{\frac{l}{\alpha}})$ space and terminates in $\tilde{O}(\sqrt{\frac{l}{\alpha}})$ passes. In particular, if l is $O(n)$ and $\alpha = n^{-\frac{1}{3}}$, the space complexity is $\tilde{O}(n^{\frac{2}{3}})$ and the number of passes required is $\tilde{O}(n^{\frac{2}{3}})$ w.h.p.*

Proof. W.h.p., Subphase (1.1) and Subphase (1.2) use $\tilde{O}(\alpha n)$ units of space. By Lemmas 2 and 4, w.h.p. $|S|$ is at most $\tilde{O}(\frac{l}{\alpha w})$. Since each node in S uses $O(1)$ space, the total space required is $\tilde{O}(\alpha n + \frac{l}{\alpha w})$ w.h.p.

Subphase (1.1) uses one pass, and Subphase (1.2) uses w passes. In the subroutine Streaming-H of Phase 2, there are originally two cases where we need one pass to sample s edges for nodes in S.

Case 1. L_u visits a stuck node $x \notin T$ and $x \notin S$. (This is the case where the random walk leaves S but visits a non-connector.)

Case 2. L_u keeps visiting vertices in S and gets trapped in S. (see page 14 for the definition of trap)

In the first case, one pass is used to get all the edges for each node in S. The number of times that Case 1 happens is bounded by $|S|$, which is $\tilde{O}(\frac{l}{\alpha w})$ w.h.p. However, Case 2 never happens in the Streaming-SS algorithm on regular graphs. Because all edges for each vertex are stored in S, regardless of how many times L_u visits a node $v \in S$, we can extend L_u using an edge that is uniformly sampled from all outgoing edges of v. Therefore, the total number of passes required in Phase 2 is $\tilde{O}(\frac{l}{\alpha w})$ w.h.p.

Adding the number of passes required in Phase 1, the total number of passes required by the Streaming-SS algorithm on regular graphs is $\tilde{O}(w + \frac{l}{\alpha w})$ w.h.p. Set $w = \sqrt{\frac{l}{\alpha}}$, and Lemma 15 is obtained.

□

The Sampling-SAS algorithm

We first introduce some modifications to the Sufficient-Sampling used by the Sampling-SAS algorithm. The modified Sufficient-Sampling keeps querying the oracle until for each vertex k distinct edges have been stored during the Normal-sampling step. Through this modification the event R can be avoided since we have stored all edges of each vertex. When uniform edge samples for a vertex v are needed in Subphase (1.2) or Subphase (2.1), we can generate as many edge samples as required by uniformly sampling all distinct edges starting from v. The modified algorithm is called Sampling-SASR, which is a Las Vegas algorithm, because its output is always correct. The same modifications can be applied to other graphs as long as the degree of each vertex is known. The next lemma describes the performance of Sampling-SASR.

Lemma 16. *Given a k-regular connected graph with n vertices, the Sampling-SASR algorithm always outputs the endpoint for a random walk of length l. W.h.p., it uses $\tilde{O}(\alpha n + \sqrt{\frac{l}{\alpha}})$ space and terminates in $\tilde{O}(\sqrt{\frac{l}{\alpha}})$ number of rounds. In particular, if $l = O(n)$ and $\alpha = n^{-\frac{1}{3}}$, the space complexity is $\tilde{O}(n^{\frac{2}{3}})$ and the number of rounds required is $\tilde{O}(n^{\frac{2}{3}})$ w.h.p.*

Proof. Using similar analysis for the Streaming-SS algorithm, Sampling-SASR requires $\tilde{O}(\alpha n + \frac{l}{\alpha w})$ space and $\tilde{O}(w + \frac{l}{\alpha w})$ number of rounds w.h.p. By setting $w = \sqrt{\frac{l}{\alpha}}$, Lemma 16 is obtained. □

Note that Sampling-LS is not used for regular graphs since the Sampling-SASR algorithm is a Las Vegas algorithm with better performance.

5.2 Random Graph

Random graphs are an important type of graphs which are generated by some random process. It is believed that studying random graphs can help us gain a deeper understanding of the structure of large graphs. In this section, we study the most common random graph model, the Erdős-Rényi model [18], denoted as $G(n, p)$ where n is the number of nodes and every possible edge occurs independently with probability p $(0 < p < 1)$.

For random graphs, we assume $np > 1+\epsilon$ (Random graphs with this property are called supercritical random graphs) and $np = O(1)$. For random graphs under this assumption, w.h.p there is a single giant connected component, with other components having size $O(\log n)$ [10]. We only focus on performing random walks on the giant component.

Random graphs are similar to regular graphs to some extent since the degree of most nodes is close to the expected degree in a random graph. The expected degree for a node is $(n-1)p \approx np$ when n is large. By our assumption, np is a constant and thus the degree of most nodes is close to a constant. The following lemma bounds the maximal degree of a vertex in a random graph. (Bollobás [9] proved a tighter bound with more a complicated proof, but the bound below helps us simplify further computation.)

Lemma 17. *The degree of any node in $G(n, p)$ does not exceed $(1 + 4\log n)(n-1)p$ w.h.p.*

Proof. First of all, consider only one specific node u. Let X_v be 1 if the possible edge $\{u, v\}$ exists, 0 otherwise. Let X equal to $\sum_{v \in V, v \neq u} X_v$, and $E[X] = (n-1)p$ can be derived. Since X_v's are identical, independent and 0-1 random variables, by Chernoff bound,

$$P[X > (1+\delta)(n-1)p] \leq e^{-\frac{\delta^2}{2+\delta}(n-1)p} (\delta > 0). \tag{5.1}$$

Let $\delta = 4\log(n)$. Because $4\log(n)$ is greater than 2,

$$P[X > (1+\delta)(n-1)p] \leq e^{-(2\log(n))(n-1)p} = n^{-2(n-1)p}. \tag{5.2}$$

According to our assumption, $(n-1)p \geq 1$, therefore w.h.p. $X \leq (1 + 4\log n)(n-1)p$.

Then we can apply the union bound over all nodes. Let Q_r denote the event that some node has degree greater than $(1 + 4\log n)(n-1)p$. Then

$$P[Q_r] \leq n \cdot n^{-2(n-1)p} \leq n^{-1}. \tag{5.3}$$

Therefore, no node in a random graph has degree greater than $(1 + 4\log n)(n-1)p$ w.h.p. □

Since np is a constant under our assumption, the space required to store all the edges incident on any node is $\tilde{O}(1)$ w.h.p.

The Streaming-SS algorithm requires exactly the same amount of space and the same number of passes on regular graphs and random graphs. The space required is $\tilde{O}(\alpha n + \frac{l}{\alpha w})$ and the number of passes required is $\tilde{O}(w + \frac{l}{\alpha w})$ w.h.p. Note that it is possible that the degree of some node in $G(n,p)$ exceeds $(1 + 4\log n)(n-1)p$. In this case, the previous probabilistic bounds do not hold.

Because the degree of each vertex in random graphs is unknown in the edge sampling model, the Sampling-SAS algorithm cannot be converted into a Las Vegas algorithm. Sampling-SAS samples the endpoint for a random walk of length l, unless R happens. In addition, Sampling-SAS uses the same space and number of rounds as in Lemma 16 w.h.p.

The Sampling-LS algorithm also achieves better performance on random graphs, which will be elaborated in the next section.

5.3 (Undirected) Fast Mixing Graph

A fast mixing graph is a graph where random walk has a mixing time $M(\epsilon) = poly(\log n, \log(\frac{1}{\epsilon}))$. The fast mixing property can be used to improve the performance of the Streaming-S algorithm and the Sampling-LS algorithm. The improvements for the Sampling-LS algorithm on fast mixing graphs will be explained in detail next. The explanation for Streaming-S algorithm's improvements is skipped because it shares similarity with that of the Sampling-LS algorithm.

As mentioned before, π^* is the stationary distribution of a random walk. After $M(\epsilon)$ steps, the random walk visits a vertex v with probability close to $\pi^*(v)$. For any set U, the probability that the random walk leaves U after $M(\epsilon)$ steps (denoted by $\hat{p}$) is close to $\sum_{v \notin U} \pi^*(v)$. If $\hat{p}$ is no less than $\frac{1}{4}$ and $M(\epsilon)$ is small, then the random walk quickly leaves the set U (as will be explained in detail later) w.h.p.

Subphase (2.1) of Sampling-LS prepares at most s uniform edge samples for each node in S, in case the random walk L_u keeps visiting nodes in S. However, for fast mixing graphs, if S satisfies the condition that $\hat{p} \geq \frac{1}{4}$, then L_u is likely to leave S in less than s steps. Therefore, Subphase (2.1) does not have to prepare s edge samples for each node in S.

When the random walk L_u stops at a stuck node in an undirected graph, let

$$c = \sum_{x \in S} \pi^*(x) = \frac{\sum_{x \in S} d(x)}{2m}. \tag{5.4}$$

After L_u proceeds with $M(\epsilon)$ steps, the probability $\hat{p}$ that the current endpoint of L_u is not in S is (σ is the distribution of L_u's endpoint before $M(\epsilon)$ steps, and P is the transition matrix)

$$\hat{p} = \sum_{x \notin S} \sigma P^{M(\epsilon)}(x) = \sum_{x \notin S} \pi^*(x) - (\pi^*(x) - \sigma P^{M(\epsilon)}(x)). \tag{5.5}$$

It can be simply derived that

$$\hat{p} \geq \sum_{x \notin S} \pi^*(x) - |\pi^*(x) - \sigma P^{M(\epsilon)}(x)| = \sum_{x \notin S} \pi^*(x) - \sum_{x \notin S} |\pi^*(x) - \sigma P^{M(\epsilon)}(x)|. \tag{5.6}$$

By the well-known equation $\|\mu - \pi\|_{TV} = \frac{1}{2} \sum_x |\mu(x) - \pi(x)|$ and the definition of mixing time,

$$\sum_{x \notin S} |\pi(x) - \sigma P^{M(\epsilon)}(x)| \leq 2 \left\| \sigma P^{M(\epsilon)} - \pi \right\|_{TV} < 2\epsilon. \tag{5.7}$$

Therefore,

$$\hat{p} > 1 - c - 2\epsilon. \tag{5.8}$$

Thus, the probability that L_u is outside of S after $M(\epsilon)$ steps is greater than $1 - c - 2\epsilon$. We consider one iteration to be the process that L_u moves for $M(\epsilon)$ steps. After each iteration the probability that L_u is outside of S is $\hat{p}$ regardless of the initial distribution of L_u in this iteration. Next, it will be shown that if $\hat{p} \geq \frac{1}{4}$, L_u leaves each set S within $\log(nl)$ iterations w.h.p. In fast mixing graphs, since $M(\epsilon)\log(nl)$ is likely to be smaller than s, only $M(\epsilon)\log(nl)$ edge samples need to be prepared for each node in S.

As L_u grows, the size of set S also grows. Consequently, c increases and probability $\hat{p}$ decreases accordingly. In order to give a lower bound of the probability $\hat{p}$, c needs to be upper bounded so that $\hat{p} \geq \frac{1}{4}$. Let c^* be the maximum value of c throughout the execution of Sampling-LS. Because c monotonically increases, c^* equals to the value of c calculated using the largest S (denoted as S^*) when the algorithm terminates. Moreover, we define ***Condition F1*** as $\frac{\sum_{x \in S^*} d(x)}{2m} + 2\epsilon < \frac{3}{4}$. If Condition F1 is satisfied, $\hat{p} \geq \frac{1}{4}$ through the algorithm.

Here we present the core results of the analysis.

Lemma 18. *Given an undirected graph $G(V, E)$ with mixing time $M(\epsilon)$, when Sampling-LS terminates, if Condition F1 is satisfied, the random walk L_u leaves set S within $M(\epsilon)\log(nl)$ steps w.h.p.*

Proof. For set S and $\hat{p}$ during the algorithm,

$$\hat{p} > 1 - c - 2\epsilon \geq 1 - c^* - 2\epsilon. \tag{5.9}$$

Let E_i be the event that the endpoint of L_u is inside S by the end of the i-th iteration. Denote the event that random walk never leaves set S within the i-th iteration by Y_i. Since

event $Y_i \subseteq E_i$,

$$Pr[Y_i] \leq Pr[E_i]. \tag{5.10}$$

As $(\bigcap_{i=1}^{k} Y_i) \subseteq (\bigcap_{i=1}^{k} E_i)$, the probability that L_u never leaves set S after k iterations (denoted as $Pr[\bigcap_{i=1}^{k} Y_i]$) satisfies

$$Pr[\bigcap_{i=1}^{k} Y_i] \leq Pr[\bigcap_{i=1}^{k} E_i]. \tag{5.11}$$

In addition,

$$Pr[\bigcap_{i=1}^{k} E_i] = Pr[E_1] \cdot Pr[E_2|E_1] \cdot Pr[E_3|E_2 \cap E_1] ... \cdot Pr[E_k| \bigcap_{i=1}^{k-1} E_i]. \tag{5.12}$$

Since each iteration has $M(\epsilon)$ steps, by the definition of mixing time, the endpoint's distribution of L_u is close to the stationary distribution. Therefore, every $Pr[E_j| \bigcap_{i=1}^{j-1} E_i]$ can be bounded using Inequality (5.8). Consequently,

$$Pr[E_j| \bigcap_{i=1}^{j-1} E_i] \leq c^* + 2\epsilon \quad (1 \leq j \leq k). \tag{5.13}$$

Thus,

$$Pr[\bigcap_{i=1}^{k} E_i] \leq (c^* + 2\epsilon)^k. \tag{5.14}$$

By Inequalities (5.11) and (5.14),

$$Pr[\bigcap_{i=1}^{k} Y_i] \leq (c^* + 2\epsilon)^k. \tag{5.15}$$

If $k = \log(nl)$ and $c^* + 2\epsilon < \frac{3}{4}$, then

$$Pr[\bigcap_{i=1}^{k} Y_i] \leq (c^* + 2\epsilon)^k \leq \frac{1}{nl}. \tag{5.16}$$

Thus, w.h.p., L_u leaves set S within $M(\epsilon) \cdot \log(nl)$ steps. Let Q_f^* be the event that, when L_u gets stuck at some node, L_u does not leave S within $M(\epsilon) \cdot \log(nl)$ steps. Because L_u can get stuck at most l times, by the union bound $Pr[Q_f^*] \leq \frac{1}{nl} \cdot l = \frac{1}{n}$. In other words, w.h.p., L_u leaves S within $M(\epsilon) \cdot \log(nl)$ steps throughout the algorithm.

□

Space Complexity and the Number of Rounds Required

Because of Lemma 18, Subphase (2.1) only needs to prepare $M(\epsilon)\log(nl)$ edges for each vertex in S rather than up to s edges. For a fast mixing graph whose $M(\epsilon)$ is $poly\log n$, one way to use Lemma 18 is sampling $poly\log n \cdot \log(nl)(\tilde{O}(1))$ edges for each vertex in S in one round, which greatly saves space in Subphase (2.1). Another way is using $O(1)$ space to store the current endpoint of L_u and using $M(\epsilon)\log(nl)$ new rounds until L_u leaves S.

Although there are two ways to use Lemma 18, only the second way is elaborated in this section due to the similarity of analysis shared by the first and the second ways. The Las Vegas algorithm Sampling-LS is modified using the second way and is now called Sampling-LSF. The performance of Sampling-LSF is given in Lemma 19.

Lemma 19. *Given an undirected connected fast mixing graph G with n vertices and any $l \geq 0$, Sampling-LSF always samples the endpoint for a random walk of length l. W.h.p., it uses $\tilde{O}(\alpha n + \sqrt{\frac{l}{\alpha}})$ space and completes in $\tilde{O}(\sqrt{\frac{l}{\alpha}})$ rounds, unless Condition F1 is not satisfied. In particular, if $l = O(n)$ and $\alpha = n^{-\frac{1}{3}}$, the space complexity is $\tilde{O}(n^{\frac{2}{3}})$ and the number of rounds required is $\tilde{O}(n^{\frac{2}{3}})$ w.h.p.*

Proof. First of all, we bound the space complexity of the Sampling-LSF algorithm. The algorithm requires $\tilde{O}(\alpha n)$ space to store connectors and the endpoints of w-length random walks w.h.p. By Lemmas 2 and 4, $|S|$ is at most $\tilde{O}(\frac{l}{\alpha w})$ w.h.p. Since each node in S uses $O(1)$ space, the space required in Phase 2 is $\tilde{O}(\frac{l}{\alpha w})$. Therefore, the space required to perform a single random walk of length l is $\tilde{O}(\alpha n + \frac{l}{\alpha w})$ w.h.p.

Phase 1 requires $O(w)$ rounds to prepare w-length random walks. In Phase 2, when L_u stops at a stuck node, by Lemma 18, w.h.p. $M(\epsilon)\log(nl)$ $(\tilde{O}(1))$ rounds are required to let L_u leave S. After leaving S, with probability α, L_u stops at a connector and makes w progress. By Lemma 4, for all nodes in S, there are at most $\tilde{O}(\frac{1}{\alpha})$ consecutive non-connectors before a connector occurs w.h.p. Therefore, if L_u stops at $\tilde{O}(\frac{1}{\alpha})$ nodes outside S, L_u is extended by at least w steps w.h.p. Moreover, the number of rounds required for L_u to stop at such amount of nodes outside S is bounded by $\tilde{O}(\frac{1}{\alpha}) \cdot \tilde{O}(1) = \tilde{O}(\frac{1}{\alpha})$ w.h.p. Consequently, the number of rounds required in Phase 2 is bounded by $\frac{l}{w} \cdot \tilde{O}(\frac{1}{\alpha}) = \tilde{O}(\frac{l}{\alpha w})$ w.h.p. Adding the number of rounds required in Phase 1, the total number of rounds required by the algorithm is $\tilde{O}(w + \frac{l}{\alpha w})$ w.h.p.

Let $w = \sqrt{\frac{l}{\alpha}}$. We obtain Lemma 19.

□

5.3.1 Analysis for Particular Fast Mixing Graphs

In this section, we use some known results to demonstrate that random graphs generated by Erdős-Rényi model and Preferential Attachment model (or Barabási-Albert model) have the fast mixing property. Then Sampling-LSF is applied to them.

Since $M(\frac{1}{4})$ mixing time in various classes of graphs has been well studied, we set ϵ to be $\frac{1}{4}$ in order to use the known results.

Random Graphs Generated by Erdős-Rényi Model

Fountoulakis et al. [20] and Benjamini et al. [8] proved independently the following theorem which indicates the mixing time for supercritical random graph $G(n,p)$ $(np > 1)$.

Theorem 5 (Theorem 1.1 in [8])**.** *If $p = \frac{c}{n}$ where $c > 1$, then C_1, the largest component of $G(n,p)$ (the unique component of linear size) satisfies*

$$M_{C_1}(\frac{1}{4}) = \Theta(\log^2(n)), \tag{5.17}$$

here $M_{C_1}(\frac{1}{4})$ denotes the $\frac{1}{4}$-mixing time for the component C_1.

Since the mixing time for supercritical random graph is *poly* $\log n$, Sampling-LSF can be used. Next, we bound the size of S^* so that Condition F1 is satisfied, where S^* is the set S when Sampling-LSF terminates.

By Lemma 17, we find that the upper bound on the degree of any vertex in S is $(1 + 4\log n)(n-1)p$ w.h.p. Then we prove the lower bound for the number of edges of $G(n,p)$ to bound the denominator $2m$ of c.

Lemma 20. *W.h.p., the number of edges of $G(n,p)$ is at least $(1 - \frac{\log n}{\sqrt{n}})pn(n-1)$.*

Proof. There are $\frac{n(n-1)}{2}$ possible edges and each edge occurs independently with probability p. Let Y_{uv} be 1 if the possible edge (u,v) exists, 0 otherwise. Let $Y = \sum_{u\in V, v\in V, u\neq v} Y_{uv}$. Then $E[Y] = \frac{pn(n-1)}{2}$. As the Y_{uv}s are identical, independent and 0-1 random variables, Chernoff bound can be used, and

$$Pr[Y < (1-\delta)E[Y]] \leq e^{-\frac{\delta^2}{2}E[Y]} \leq e^{-\frac{\delta^2}{4}n}. \tag{5.18}$$

If we set $\delta = \frac{\log n}{\sqrt{n}}$, we get

$$Pr[Y < (1-\delta)E[Y]] \leq e^{-\frac{\log^2(n)}{4}} \leq \frac{1}{n}. \tag{5.19}$$

□

Condition F1 implies that

$$\frac{\sum_{x\in S^*} deg(x)}{2m} \leq \frac{|S^*|(1+4\log n)(n-1)p}{2\left(1-\frac{\log n}{\sqrt{n}}\right)pn(n-1)} = \frac{|S^*|(1+4\log n)}{2\left(1-\frac{\log n}{\sqrt{n}}\right)n} < \frac{1}{4}. \tag{5.20}$$

Then,

$$|S^*| < \frac{\left(1-\frac{\log n}{\sqrt{n}}\right)n}{2(1+4\log n)}. \tag{5.21}$$

As proved earlier, $|S^*| = \tilde{O}(\frac{l}{w\alpha})$. For any l, w and α can be set so that Inequality (5.21) is satisfied.

Power Law Graphs

Power law graphs are graphs whose degree distribution satisfies the power law asymptotically. That is, the fraction $P(k)$ of vertices of degree k is

$$P(k) \sim k^{-\alpha}, \tag{5.22}$$

where C is a constant and α, called *exponent*, is also a constant.

Many networks are found to have the power law property. For example, Price [16] found that the number of citations a paper receives in the citation network satisfies the power law. Barabási et al. [5] found that the number of actors that an actor has collaborated with satisfies the power law with exponent $\alpha = 2.3 \pm 0.1$. Therefore, it is important to perform random walks on power law graphs.

In order to explain the power law property in networks, Barabási et al. [5] proposed a model called preferential attachment model or Barabási-Albert model. The model uses the following procedure to generate a random power law graph $G_d(n)$.

Initially, the graph has d_0 vertices, the edges between which are chosen arbitrarily, as long as each vertex has at least one edge. Then we add n new vertices to the graph. Each new vertex connects to d existing vertices such that the probability p_i that the i-th new vertex connects to the vertex u is

$$p_i = \frac{d(u)}{\sum_{v \in V} d(v)}, \tag{5.23}$$

where $d(v)$ is the current degree for vertex v.

This process is called preferential attachment process because vertices of high degree are more likely to be connected by more edges when new nodes are added. Moreover, Barabási et al. [5] proved that $G_d(n)$ is a power law graph with exponent $\alpha = 3$.

The preferential attachment model is the first well-known model attempting to generate and explain a power law network. Although it has the limitation that it can only generate power law graphs with exponent 3, it is still worth studying.

Mixing time is inversely correlated to *conductance* in a graph. Next we use a known conductance property of the preferential attachment model to bound its mixing time. The conductance of a graph G is defined as ($\overline{S} = V \setminus S$)

$$\Phi_G = \min_{S \subseteq V} \frac{\sum_{u \in S, v \in \overline{S}} a_{uv}}{min(a(S), a(\overline{S}))}, \tag{5.24}$$

where a_{uv} is 1 if edge (u, v) exists (otherwise 0), and

$$a(S) = \sum_{u \in S} \sum_{v \in V} a_{uv} \tag{5.25}$$

is the number of edges incident to vertices in S.

Mihail et al. [36] proved the constant conductance property for preferential attachment graph $G_d(n)$ in the following theorem, in which $\rho_{G_d(n)}$ is the edge expansion of the graph and $\Phi_{G_d(n)}$ is the conductance of the graph. The edge expansion of a graph G, also known as its Cheeger constant is defined as

$$\rho_{G_d(n)} = \min_{U \subset V, 0 < |U| < \frac{n}{2}} \frac{E(U, \overline{U})}{|U|}, \tag{5.26}$$

where $E(U, \overline{U})$ denotes the set of edges with one end in U and the other end in $V \setminus U$.

Theorem 6 (Theorem 1 in [36])**.** *For every positive constant integer $d \geq 2$ and for every positive constant $c < 2(d-1) - 1$, there is a positive constant $\gamma = \gamma(d, c)$ such that the random graph $G_d(n)$ has edge expansion γ and conductance $\frac{\gamma}{d+\gamma}$ w.h.p. In particular, for $\gamma < min\{\frac{d-1}{2} - \frac{c+1}{4}, \frac{1}{5}, \frac{(d-1)\ln 2 - \frac{2}{5}\ln 5}{2(\ln d + \ln 2 + 1)}\}$,*

$$Pr[\rho_{G_d(n)} < \gamma] \leq o(n^{-c}), \tag{5.27}$$

and

$$Pr[\Phi_{G_d(n)} \leq \frac{\gamma}{d+\gamma}] \leq o(n^{-c}). \tag{5.28}$$

Paper [36] proved the constant conductance property of preferential attachment graphs but did not bound mixing time. Now, we use the inequality below to bound the mixing time for preferential attachment graphs. The inequality (Φ is the conductance of the graph and M is the mixing time.),

$$\Phi^2/2 \leq \frac{1}{M} \leq 2\Phi, \tag{5.29}$$

was first proven by Jerrum et al. [23].

We let $M_{G_d(n)}$ be the mixing time for the preferential attachment graph, then

$$M_{G_d(n)} \leq \frac{2}{\Phi_{G_d(n)}{}^2}. \tag{5.30}$$

By Theorem 6, w.h.p.

$$M_{G_d(n)} \leq 2\left(\frac{d+\gamma}{\gamma}\right)^2. \tag{5.31}$$

Since the mixing time of the preferential attachment graph can be bounded by a constant w.h.p., our algorithms for fast mixing graphs can be applied to preferential attachment graphs.

The Sampling-LSF algorithm can also be used on other power law graphs as long as the graphs have the fast mixing property.

Now for fast mixing power law graphs with exponent α, we bound the maximal size of S^* so that Condition F1 is satisfied.

Recall that Condition F1 requires that $\frac{\sum_{x \in S^*} d(x)}{2m} < \frac{1}{4}$ ($\epsilon = \frac{1}{4}$). In the worst case, S^* contains as many vertices of high degree as possible. Thus, we need to find out the smallest fraction of vertices whose degree sum is half of the graph's total degree since this fraction is the upper bound of $|S^*|$.

We assume the degree's power law distribution is

$$p(x) = Cx^{-\alpha} \tag{5.32}$$

with $\alpha > 0$ and $x \in \mathbb{N}^+$. Although the degree distribution of a power law graph is asymptotically close to Equation (5.32) by definition, we use Equation (5.32) to roughly calculate the bound of $|S^*|$.

First of all, we calculate the constant C. As

$$1 = \sum_{x \in \mathbb{N}^+} p(x), \tag{5.33}$$

$$C = \frac{1}{\sum_{x \in \mathbb{N}^+} x^{-\alpha}}. \tag{5.34}$$

The sum $\sum_{x \in \mathbb{N}^+} x^{-\alpha}$ is the Riemann zeta function, and it is denoted by $\zeta(\alpha)$. Thus,

$$p(x) = \frac{x^{-\alpha}}{\zeta(\alpha)}. \tag{5.35}$$

Let $P(x)$ be the probability that the degree of a node is greater than x, then

$$P(x) = \sum_{t > x, t \in \mathbb{N}^+} \frac{t^{-\alpha}}{\zeta(\alpha)}. \tag{5.36}$$

In addition, let $W(x)$ be the fraction of the total degree of vertices that have degree more than x. Thus,

$$W(x) = \frac{\sum_{t > x, t \in \mathbb{N}^+} t \frac{t^{-\alpha}}{\zeta(\alpha)}}{\sum_{t \in \mathbb{N}^+} t \frac{t^{-\alpha}}{\zeta(\alpha)}} = \frac{\sum_{t > x, t \in \mathbb{N}^+} t^{-\alpha+1}}{\zeta(\alpha - 1)}. \tag{5.37}$$

Since

$$\int_k^{k+1} t^{-\alpha} dt \leq k^{-\alpha} \leq \int_{k-1}^{k} t^{-\alpha} dt, \tag{5.38}$$

then

$$W(x) \leq \frac{1}{\zeta(\alpha - 1)} \int_x^{\infty} t^{1-\alpha} dt = \frac{1}{\zeta(\alpha - 1)} \frac{x^{2-\alpha}}{\alpha - 2}. \tag{5.39}$$

In order to satisfy Condition F1, we require that

$$W(x) \leq \frac{1}{\zeta(\alpha-1)} \frac{x^{2-\alpha}}{\alpha-2} < \frac{1}{4}, \tag{5.40}$$

and thus

$$x > \left(\frac{\zeta(\alpha-1)(\alpha-2)}{4} \right)^{\frac{1}{2-\alpha}}. \tag{5.41}$$

Let $\hat{x} = \left(\frac{\zeta(\alpha-1)(\alpha-2)}{4} \right)^{\frac{1}{2-\alpha}}$. Then the maximal size of S^* is bounded by $P(\hat{x})$.

Chapter 6

Applications

In this chapter, we first develop an algorithm called Sampling-AM that samples the endpoints of multiple random walks based on Sampling-AS. Then we use Sampling-AS to estimate PageRank scores for vertices. Sampling-AM is further modified to sample the endpoints of absorbing random walks and it is used in a random walk based recommendation system.

6.1 Multiple Random Walks

Applications in Sections 6.2 and 6.3 involve the endpoints of a large number of random walks. We can either use our Sampling-AS algorithm to sample their endpoints one after another or execute multiple instances of Sampling-AS in parallel.

If multiple instances of Sampling-AS execute Sufficient-Sampling in parallel, an edge sample drawn in the Normal-sampling step might be used more than once. Thus, these multiple random walks are not independent. Therefore, when parallel executions of Sufficient-Sampling are needed, we use a technique called Parallel-Sufficient-Sampling that generates all edge samples required together. Assume K random walks are being constructed. Let U_1 denote the current distinct endpoints of K random walks, then for each node $u \in U_1$, Parallel-Sufficient-Sampling counts $m_u{}'$ which is the number of edge samples starting from u required by all K random walks. Next, Parallel-Sufficient-Sampling uses Sufficient-Sampling with $U = U_1$ and $m_u = m_u{}'$ for each $u \in U_1$. In this way, all edge samples generated by Parallel-Sufficient-Sampling are never reused; therefore, all K random walks are independent. The space complexity is the same as that required by K instances of Sufficient-Sampling since the number of edge samples stored does not change. Moreover, the error probability is the same as that for a single execution of Sufficient-Sampling since the "bad" event is still R^*.

Hence, executing K instances of Sampling-AS in parallel (using Parallel-Sufficient-Sampling) requires $\tilde{O}(K\alpha n + K\frac{l^{\frac{1}{2}}}{\alpha})$ space w.h.p. The number of rounds required is still $\tilde{O}(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha})$ w.h.p., because each execution of Sampling-AS can use the same round.

Paper [42] uses an elegant technique to reduce the space required by multiple random walks. Since the technique can be directly used in our solution, we will briefly describe the technique.

The main idea of the technique is to reduce the number of w-length random walks we prepare. When we perform multiple random walks, some connectors are likely to be used by most of the random walks while others are not likely to be used. Therefore, instead of preparing K w-length random walks for each connector, we prepare w-length random walks based on the probability p_v that the w-length random walks of a connector v is used. This technique can save much space when K is large. Moreover, if we use K^* random walks and count n_v^*, the number of w-length random walks of the connector v that is used, then we can estimate p_v using $\frac{n_v^*}{K^*}$. We start by using a small K^* and then double the number of random walks each time to get a more accurate estimation of p_v.

In order to estimate p_v, we need the following lemma.

Lemma 21 (Lemma 4.2 in [42])**.** *If the probability of an event X occurring is p, then in $t = \Theta(\log n/\epsilon)$ trials, the fraction of times the event X occurs satisfies*

$$p - \sqrt{p\epsilon} - \epsilon \leq X \leq p + \sqrt{p\epsilon} + \epsilon, \tag{6.1}$$

w.h.p.

We omit the proof for the lemma here since it can be found in [42]. Note that when $t = 8(\log(nKl)/\epsilon) = O(\log n/\epsilon)$, this process succeeds with probability at least $1-(nKl)^{-2}$. Also, if $p \gg \epsilon$, the approximation ratio is close to 1. Next we will give the precise definition for p_v.

Definition 6.1.1 (Definition 4.4 in [42])**.** *For every connector v, p_v is defined as the probability that during the execution of the algorithm Streaming-S, the w-length walk of v is used (and hence v gets included in the set S).*

We call our algorithm Sampling-AM, which samples the endpoints of multiple random walks in the edge sampling model. The algorithm runs in $\log K$ phases and in Phase $j+1$ we will use $O(2^j \log n)$ random walks in parallel to estimate p_v with an additive error of $\sqrt{p_v/2^j} + 1/2^j$ (we denote the estimated p_v by $\widetilde{p_v}$). This $\widetilde{p_v}$ is then used in Phase $j+2$ to calculate the number of w-length random walks we need to prepare for each connector. After w-length random walks are prepared, in Phase $j+2$, the algorithm spawns $O(2^{j+1} \log n)$ instances of Sampling-AS using Parallel-Sufficient-Sampling. (Note that Phase 1 processes of all instances of Sampling-AS have been completed.) Please refer to the pseudo code for details.

The performance of Sampling-AM is stated in Theorem 7. "Bad" event R_M is the event that R (see page 33) happens in some execution of Sampling-AS.

Theorem 7. *Given a graph G with n vertices, any $l \geq 0$ and any non-negative K upper bounded by poly(n), Sampling-AM samples the endpoints of K l-length random walks, unless the event R_M happens. (R_M happens with probability at most $\tilde{O}(\frac{1}{n})$.) The algorithm terminates in $\tilde{O}(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha})$ rounds and uses $\tilde{O}(\alpha n + K\frac{l^{\frac{1}{2}}}{\alpha})$ space with probability at least $1 - O(\frac{1}{n})$, for any choice of α with $0 < \alpha \leq 1$.*

Algorithm 11 Sampling-AM(σ, l, K)

Input: Initial distribution σ, length of the walk l and the number of random walks K
$T \leftarrow$ nodes obtained by sampling each node independently with probability α.
Perform phases 1 through log K as follows.
Phase 1:
    Sample the endpoints of $O(\log n)$ walks of length w starting from each connector in w rounds. These w-length random walks are sufficient for $O(\log n)$ parallel executions of Sampling-AS.
    Spawn $K_1 = O(\log n)$ instances of Sampling-AS (using Parallel-Sufficient-Sampling) to obtain K_1 walks. All these instances only use the w-length walks generated in the previous step.
    For each connector v, use $\widetilde{p_v} = n_v/K_1$ to estimate p_v, where n_v is the number of used w-length walks of v generated in the current phase.
Phase (j+1): {The estimated $\widetilde{p_v}$ is known up to an additive error of $\sqrt{p_v/2^{j-1}}+1/2^{j-1}$ }
    In w rounds, find the endpoints of $O(2^j\widetilde{p_v}\log n + \log n)$ random walks of length w starting from connector v. These w-length random walks are sufficient for $O(2^j \log n + \log n)$ parallel executions of Sampling-AS.
    Run $K_{j+1} = O(2^j \log n + \log n)$ instances of Sampling-AS using the w-length random walks sampled in the previous step. (When parallel executions of Sufficient-Sampling is needed, Parallel-Sufficient-Sampling is used.)
    Estimate the p_v using $\widetilde{p_v} = n_v/K_{j+1}$ where n_v is the number of used w-length walks of v generated in the current phase. This estimate is accurate up to an additive error of $\sqrt{p_v/2^j} + 1/2^j$ w.h.p. (by Chernoff bound).

Proof. We show that the number of w-length random walks in each phase is sufficient to perform single random walks in that phase w.h.p. Since we perform $O(2^{j-1}\log n + \log n)$ random walks in Phase j, after Phase j, our estimated $\widetilde{p_v}$ satisfies $\widetilde{p_v} \geq p_v - \sqrt{p_v/2^{j-1}} - 1/2^{j-1}$ w.h.p by Lemma 21 with $\epsilon \leq 2^{-(j-1)}$. Then we can deduce $p_v \leq 2\widetilde{p_v} + \frac{1}{2^{j-1}} = O(\widetilde{p_v} + \frac{1}{2^{j-1}})$ w.h.p. Since we generate K_{j+1} random walks in Phase $j+1$, by Lemma 21 with $\epsilon \leq 2^{-j}$, the number of random walks that use connector v's w-length random walks is at most $K_{j+1}(p_v + \sqrt{p_v/2^j} + 1/2^j) \leq O(2^j\widetilde{p_v}\log n + \log n)$. This is exactly the number of w-length random walks we prepare in Phase $j+1$.

In Phase 1, we start with $8\log(nKl)$ random walks. Therefore, in each phase, we use at least $8\log(nKl)$ random walks to estimate p_v. Hence, each estimation for p_v satisfies Inequality (6.1) with probability at least $1-(nKl)^{-2}$. We assume that $\alpha \cdot \log K \cdot \log n \leq K$, then the total number of estimations for p_v is at most $\log K \cdot |T| \leq Kn$ ($|T|$ is the number of connectors). Consequently, the probability that some estimation $\widetilde{p_v}$ does not satisfy Inequality (6.1) during the Sampling-AM is at most $(nKl)^{-2} \cdot Kn = \frac{1}{nKl^2}$ by the union bound. Hence, the number of w-length random walks in each phase is sufficient to execute the instances of Sampling-AS in the same phase w.h.p.

We need $\tilde{O}(\alpha n)$ units of space to store connectors. Then we need to store the endpoints of w-length random walks in Phase $t+1$ using $O(\sum(2^t\widetilde{p_v}\log n + \log n))$ of space. In addition, we observe that $\sum p_v \leq \frac{l}{w}$. Since the last phase stores the most w-length random walks, the space used to store w-length random walks is at most $O(\sum(K\widetilde{p_v}\log n + \log n)) = \tilde{O}(K\frac{l}{w} + \alpha n)$.

Let Q_0 denote the event that some node is not considered when sampling connectors, then $Pr[Q_0] \leq \frac{1}{n}$. In addition, let Q_1 denote the event that the number of connectors is $\tilde{O}(\alpha n)$, then $Pr[Q_1] \leq \frac{1}{n}$. By Lemma 10, if Q_0 does not happen, the probability that there are more than i consecutive non-connectors before a connector occurs (denoted as Q_2) is at most $\frac{1}{nl}$, when $i = \frac{\log(nl)}{\alpha}$. In Sampling-AM, we increase i from $\frac{\log nl}{\alpha}$ to $\frac{\log(nK\log(K)l^2)}{\alpha}$,then $Pr[Q_2] \leq \frac{1}{nK\log Kl^2} \cdot |S| \leq \frac{1}{nK\log(K)l}$ by the union bound. Let $Q_2{}'$ be the event that Q_2 does not happen for all random walks in Sampling-AM, then $Pr[Q_2{}'] \leq K\log K \cdot \frac{1}{nK\log(K)l} = \frac{1}{nl}$ by the union bound (the total number of random walks the algorithm constructs is bounded by $K\log K$). Q_M denotes the union of Q_0, Q_1 and $Q_2{}'$ and $Pr[Q_M] \leq \frac{3}{n}$. Since the increased value of i is still $\tilde{O}(\frac{1}{\alpha})$, Lemma 10 holds with even higher probability. By doing so, the space required in Subphase (2.1) of Sampling-LS is $\tilde{O}(K\frac{ls}{\alpha w})$, unless Q_M happens. Therefore, the space complexity of Sampling-AM is $\tilde{O}(K\frac{ls}{\alpha w} + \alpha n)$ w.h.p.

The number of rounds needed in any phase is the same as the Sampling-AS algorithm. Since we have $\log K$ phases, the total number of rounds needed is $\tilde{O}(w + \frac{l}{\alpha w} + \frac{l}{s})$ w.h.p.

Now we bound the error probability. As R_M is equivalent to the event that R does not happen in $\log K$ instances of Sampling-AS. By the union bound, $Pr[R_M] \leq \log K\frac{1}{n} = \tilde{O}(\frac{1}{n})$.

Let $w = l^{\frac{3}{4}}$ and $s = l^{\frac{1}{4}}$, Theorem 7 is obtained. □

6.2 Estimating PageRank Scores

PageRank is the main problem solved by [42] for the graph streaming model. Here, we briefly describe the idea and also apply a similar idea to the edge sampling model.

The PageRank vector can be considered as the stationary distribution of a special random walk (we call it the PageRank random walk). For each step, this random walk L_u proceeds as a regular random walk with probability β, and with probability $(1-\beta)$ L_u moves to a uniformly selected node (we call such a step *jump* and a jump is still consid-

ered as one step). L_u keeps moving until the length of L_u reaches the value needed. The PageRank score $PR(v)$ is defined using the following equation ($N(v)$ denotes the neighbors of node v)

$$PR(v) = \frac{1-\beta}{n} + \beta \sum_{u \in N(v)} \frac{PR(u)}{d(v)}. \tag{6.2}$$

For PageRank random walks, we let $M(\delta)$ denote the mixing time and let π^* denote the stationary distribution. Since $PR(v) = \pi^*(v)$, $PR(v)$ can be estimated using the endpoints of many PageRank random walks with length equal to the value of $M(\delta)$. Paper [42] estimates the PageRank vector of a graph G by sampling the endpoints of $M(\delta)$-length random walks on a modified graph G'. G' is constructed from graph G by adding a complete graph on it.

Although in the graph streaming model, we can change the edge streams to modify the original graph, we cannot modify the oracle in the edge sampling model. Therefore, we need to sample the endpoints of $M(\delta)$-length PageRank random walks using the oracle that uniformly outputs an edge of the original graph G per query.

6.2.1 Single PageRank Random Walk

Because the PageRank random walk jumps to a uniform vertex with probability $1-\beta$, a technique called Node-Sampling is developed to uniformly sample k nodes with repetition from G. Now we describe Node-Sampling in detail.

We still assume all vertices are pre-labeled with distinct labels. We randomly pick k independent hash functions that uniformly map the labels of the nodes to the interval $(0, 1)$ and let f_i denote the i-th hash function.

Then Node-Sampling uses two rounds of edge samples. For each hash function f_i, Node-Sampling uses a variable V_i to store the node with the highest value hashed by f_i. For each edge sample e, its two ends are hashed using all the hash functions and the V_is are updated when necessary. Finally, Node-Sampling outputs all the V_is.

Let R^* denote the event that some edge in G is not returned by the oracle during Node-Sampling. (R^* denotes the same event in Sufficient-Sampling of Sampling-AS.) Vertices in the V_is are uniform node samples with repetition, unless the event R^* happens.

Lemma 22. *Given a graph $G(V, E)$ and any $k > 0$ in the edge sampling model, Node-Sampling uniformly and independently samples k nodes with repetition from all the nodes of V, unless the event R^* happens. The probability that R^* happens is at most $n^{-2}l^{-1}$. In addition, Node-Sampling always uses two rounds and $O(k)$ space,*

Proof. If event R^* does not happen, then every node is considered during Node-Sampling. Since each node has the same probability to be hashed to the highest value by every hash function, each node is sampled with the same probability. Moreover, since hash functions are

independent, the samples in the V_is are also independent. In addition, the space complexity of Node-Sampling is $O(k)$ because the V_is only store k vertices.

Finally, by Lemma 5, the probability that R^* happens is at most $n^{-2}l^{-1}$. □

Now, we modify the Sampling-AS algorithm to sample the endpoint of a single $M(\delta)$-length PageRank random walk. We call the modified algorithm Sampling-ASP and its subroutine Sampling-AHP.

Subphase (1.2) is again divided into w iterations and L_i denotes the i-th random walk. Within a specific iteration, for each random walk L_i, with probability β, L_i is put into a set A. Otherwise L_i is put into a set B. Thus, A contains random walks that will move regularly and B contains random walks that will jump. Then for random walks in A, we use Sufficient-Sampling as before to extend them, while for random walks in B, we use Node-Sampling to sample $|B|$ uniform node samples with repetitions to extend them. Since Sufficient-Sampling and Node-Sampling can be executed in parallel, the number of rounds required in Subphase (1.2) is still $O(w)$ rounds.

The pseudocode for Sampling-AHP is shown below.

Algorithm 12 Sampling-AHP(x, T, S, l, L_u)

while $|L_u| < l$ **do**
  Subphase (2.1) Sample s edges (with repetition) for every node $u \in S$ into E_u and also uniformly sample s nodes into W. (using two rounds)
  Extend L_u using edges sampled in Subphase (2.1). (When visiting a node $v \in S$ for the k-th time, with probability β, use the k-th sampled edge from E_v. With probability $1 - \beta$, use the k-th sampled node in W.)
  $x \leftarrow$ new end point after extending L_u (Then there are three cases.)
  (1) if $(x \in S)$, **continue** (L_u is still in S, at least s progress is made.)
  (2) if ($x \in T$ and $x \notin S$), **return** (This means L_u reaches an unused connector and we can use it to extend L_u.)
  (3) if ($x \notin T$ and $x \notin S$), $S \leftarrow S \cup \{x\}$, **continue** ($L_u$ reaches a new non-connector.)
end while
return the endpoint of L_u

In Sampling-AHP, Subphase (2.1) executes Sufficient-Sampling and Node-Sampling in parallel using two rounds. Sufficient-Sampling draws s edge samples for each vertex v in S (storing them in the multiset E_v) and Node-Sampling draws s uniform node samples with repetition (storing them in the multiset W). When the PageRank random walk L_u visits a node $v \in S$ for the k-th time, with probability β, L_u moves regularly and uses the k-th edge in E_v. With probability $1 - \beta$, L_u jumps and its endpoint is set to the k-th node in W. The rest of the procedures are the same as those in Sampling-AH.

Performance and Error Bounds

First of all, we bound space complexity. The space required to store connectors and the endpoints of w-length random walks is $\tilde{O}(\alpha n)$ w.h.p. By Lemma 10, $|S|$ is at most $\tilde{O}(\frac{l}{\alpha w})$ w.h.p. Subphase (2.1) stores s edges for each node in S and s nodes in total. Therefore, Subphase (2.1) requires $\tilde{O}(\frac{ls}{\alpha w})$ space w.h.p. Together, the space complexity of Sampling-ASP is $\tilde{O}(\alpha n + \frac{ls}{\alpha w})$ w.h.p.

Since we prepare s outgoing edges for each vertex in S and s uniform node samples with repetition, if L_u keeps visiting vertices in S, L_u will still make at least s progress. The rest of the analysis is the same as that for Sampling-AS, and we get the same bound $\tilde{O}(w + \frac{l}{\alpha w} + \frac{l}{s})$ on the number of rounds required w.h.p.

Finally, as Node-Sampling and Sufficient-Sampling have the same "bad" event R^*, Sampling-ASP shares the same error bounds with Sampling-AS. Sampling-ASP correctly samples the endpoint of a single PageRank random walk, unless R happens (R happens with probability at most $\frac{2}{n}$).

6.2.2 Multiple PageRank Random Walk

Now, we are going to use Sampling-AM to sample the endpoints of a large number of PageRank random walks. Instead of using Sampling-AS as a subroutine for Sampling-AM, we use Sampling-ASP. Since Sampling-ASP and Sampling-AS have the same performance and error bounds, Theorem 7 also holds for multiple PageRank random walks. We estimate the PageRank score by sampling the endpoints of $N = 8\frac{1}{\epsilon}\log(nM(\delta))(O(\frac{1}{\epsilon}\log n))$ PageRank random walks of length $M(\delta)$ and count the number of times n_v that node v occurs as an endpoint. Let $\widetilde{PR}(v)$ denote the estimated value of $PR(v)$, then $\widetilde{PR}(v) = \frac{n_v}{N}$. By the definition of mixing time, the probability that the random walk ends at v after $M(\delta)$ steps is close to $PR(v)$ with an absolute error less then δ. If event R_M (see page 58) does not happen, by Lemma 21, using N number of random walks, we can get

$$PR(v) - \sqrt{PR(v)\epsilon} - \epsilon - \delta \leq \widetilde{PR}(v) \leq PR(v) + \sqrt{PR(v)\epsilon} + \epsilon + \delta \tag{6.3}$$

with probability at least $1 - n^{-2}$. Therefore, if R_M does not happen and $PR(v)$ is large, w.h.p. $\widetilde{PR}(v)$ is a good estimate since the relative error is small.

By Theorem 7, the total number of rounds required is $\tilde{O}(M(\delta)^{\frac{3}{4}} + \frac{M(\delta)^{\frac{1}{4}}}{\alpha})$ w.h.p. The space complexity is $O(n)$ since we need to use a counter to count n_v for each $v \in V$.

Lastly, we bound the error probability for this procedure. Let X denote the event that the PageRank estimation $\widetilde{PR}(v)$ satisfies Inequality (6.3), then

$$Pr[X|\overline{R_M}] \geq 1 - n^{-2}. \tag{6.4}$$

By Theorem 7,

$$Pr[\overline{R_M}] \geq 1 - \tilde{O}\left(\frac{1}{n}\right). \tag{6.5}$$

Since $X = (X \cap \overline{R_M}) \cup (X \cap R_M)$,

$$Pr[X] = Pr[X|\overline{R_M}] \cdot Pr[\overline{R_M}] + Pr[X|R_M] \cdot Pr[R_M]] \geq Pr[X|\overline{R_M}] \cdot Pr[\overline{R_M}]. \tag{6.6}$$

By the three inequalities above,

$$Pr[X] \geq (1 - n^{-2}) \cdot \left(1 - \tilde{O}\left(\frac{1}{n}\right)\right) \geq 1 - \left(n^{-2} + \tilde{O}\left(\frac{1}{n}\right)\right) \geq 1 - \tilde{O}\left(\frac{1}{n}\right). \tag{6.7}$$

Therefore, using Sampling-AM to estimate PageRank scores succeeds w.h.p. Compared to the trivial algorithm that uses $O(n)$ space and $O(M(\delta))$ rounds, the Sampling-AM algorithm reduces the number of rounds required to $\tilde{O}(M(\delta)^{\frac{3}{4}} + \frac{M(\delta)^{\frac{1}{4}}}{\alpha})$ w.h.p.

6.3 Trust-based Recommendation Systems

Recommendation systems are useful in the modern society. For instance, Amazon uses a recommendation system to recommend products to users; Netflix recommends movies based on its recommendation system; Apple iTunes provide recommendations on music using a recommendation system.

Although there are multiple categories of recommendation systems, we focus on trust-based systems. Informally speaking, the problem is to estimate the recommendation rating of a single item for some specific user, given the network in which a few users have already rated the item. (Users are represented by nodes.) In daily life, a user is likely to ask people he trusts for opinions. In trust-based recommendation systems, such trust relations between people are represented by edges.

Andersen et al. [2] modeled the above problem using an annotated directed graph called a *voting network*. Users that have rated the item are called *voters*. To make the problem simpler, voters are assumed to have only two ratings - either recommend the item (labeled with $+$) or do not recommend the item (labeled with $-$). A directed edge from node a to node b means that a trusts b. Then we need to provide recommendation for a given user s (an unlabeled node).

To be precise, we give the formal definitions for a voting network and the recommendation problem.

Definition 6.3.1. *A voting network is a directed unweighted annotated multigraph $G = (N, V_+, V_-, E)$ where N is a set of nodes, V_+, $V_- \subseteq N$ are disjoint subsets of positive and negative voters, and $E \subseteq N^2$ is a multiset of edges with parallel edges allowed and no self-loops.*

When V_+ and V_- are clear from context, we denote the set of voters by $V = V_+ \cup V_-$ and the set of non-voters by $\overline{V} = N \setminus V$.

Definition 6.3.2. *A recommendation system $\hat{R}$ takes a voting network G and source $s \in \overline{V}$ as input and outputs recommendation $\hat{R}(G, s) \in \{-, 0, +\}$.*

The output + shown above means that $\hat{R}$ recommends the item; the output $-$ means that $\hat{R}$ does not recommend the item; the output 0 represents that $\hat{R}$ has no opinion. In addition, $sgn : \mathbb{R} \to \{-, 0, +\}$ denotes the function that computes the sign of its input.

6.3.1 Random Walk Recommendation System

One good heuristic algorithm that solves the recommendation problem uses random walks.

The random walk recommendation algorithm starts a random walk at the source node s. Upon visiting a node v, the random walk uniformly samples an outgoing edge of v and then uses the edge to proceed. Let Z denote the set that contains all non-voters without outgoing edges; the random walk keeps going until it visits a voter (a node in V), or a node in Z. Let p_s be the probability that the random walk starting at s terminates at a node with a positive vote and q_s be the probability that the random walk starting at s terminates at a node with a negative vote. Let $r_s = p_s - q_s$. The random walk recommendation system recommends $sgn(r_s)$ to s.

By sampling the endpoints of a large number of random walks starting from s, p_s and q_s can be estimated. Thus, r_s can also be estimated.

Absorbing Random Walks

Since random walks will stop when visiting nodes in $Z \cup V$, these random walks can be considered *absorbing random walks*. Absorbing random walks are random walks with *absorbing nodes*. Once a random walk reaches such a node, the random walk stops. Therefore, we need to modify Sampling-AM to sample the endpoints of absorbing random walks.

Another important point worth mentioning is that we do not know after how many steps a random walk starting from s will stop. Therefore, normally, we set a maximum length l for these random walks and force a random walk to stop and output 0 when it reaches l steps. This maximum length will cause some estimation error for r_s and will be further explained later.

Since a voting network is annotated, we assume every edge e returned by the oracle in the edge sampling model also contains labels so that the labels for both ends of e are known.

First of all, we modify the procedure that prepares w-length random walks. Since these random walks might visit absorbing nodes before they reach length w, we call these random walks short random walks. Whenever a short random walk visits a node, we check whether

the node is an absorbing node or not. If it is, the short random walk stops and this absorbing node becomes the endpoint. The probabilistic bounds on space complexity and the number of rounds required do not change.

Then we start random walks from the source node s and follow similar procedures as in Sampling-AM. If a random walk visits a connector with an unused short random walk, the random walk will move to the endpoint of this short random walk. We terminate a random walk and output the label of its endpoint if the random walk visits an absorbing node; otherwise, we let the random walk continue. In the case where a random walk stops at a stuck node, we use the subroutine Sampling-AH that samples s edges for each node in S, with the exception that after each step, we will check whether the current node of the random walk is an absorbing node (if so, terminate the random walk and output its endpoint) or not (continue the random walk).

The following theorem states the core results of our estimation for r_s. The proof is given in the next two sections.

Theorem 8. *Given a voting network G with n vertices, an unlabeled node s, any integer $l \geq 0$ and any positive $\epsilon \leq 1$, by sampling the endpoints of $\hat{N} = 8\frac{1}{\epsilon}\log(nl)$ absorbing random walks with length at most l starting from s, r_s can be estimated by $\widetilde{p_s} - \widetilde{q_s}$, where $\widetilde{p_s} = \frac{p_n}{\hat{N}}$ (p_n is the number of absorbing random walks that terminate at positive voters) and $\widetilde{q_s} = \frac{q_n}{\hat{N}}$ (q_n is the number of absorbing random walks that terminate at negative voters). Moreover, let $\widetilde{r_s} = \widetilde{p_s} - \widetilde{q_s}$; then w.h.p., the absolute error of the estimation is bounded by*

$$|r_s - \widetilde{r_s}| \leq \sqrt{2\left(\widetilde{p_s} + \frac{3}{2}\epsilon + \beta\right)\epsilon} + \sqrt{2\left(\widetilde{q_s} + \frac{3}{2}\epsilon + \beta\right)\epsilon} + 2\epsilon + \beta, \tag{6.8}$$

where β is the fraction of random walks that terminate at length l. In addition, the estimation terminates in $\tilde{O}(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha})$ rounds and uses $\tilde{O}(\alpha n + \frac{1}{\epsilon}\frac{l^{\frac{1}{2}}}{\alpha})$ space w.h.p.

The Space Complexity and the Number of Rounds Required

Obviously, the algorithm that samples the endpoints of multiple absorbing random walks has the same space complexity as the Sampling-AM algorithm.

In terms of the number of rounds required, if a random walk terminates at the maximum length l, the number of rounds required for it is the same as the Sampling-AM algorithm. However, if a random walk terminates before length l, it must either terminate right after using a short random walk or while handling a stuck node. In both cases, before using the last short random walk or before the last invocation of the subroutine to handle stuck node, all previous steps for the random walk are generated exactly the same way as when constructing a regular random walk without absorbing nodes, as otherwise the random walk would have stopped earlier. Therefore, the number of rounds required is at most the same as the number of rounds required by Sampling-AM.

Therefore, our modified multiple random walk algorithm that samples the endpoints of multiple absorbing random walks has the same bounds on space complexity and the number of rounds as the original Sampling-AM algorithm. By Theorem 7, the estimation terminates in $\tilde{O}(l^{\frac{3}{4}} + \frac{l^{\frac{1}{4}}}{\alpha})$ rounds and uses $\tilde{O}(\alpha n + \frac{1}{\epsilon}\frac{l^{\frac{1}{2}}}{\alpha})$ space w.h.p.

Absolute Error

We use $\hat{N} = 8\frac{1}{\epsilon}\log(nl) = O(\frac{1}{\epsilon}\log n)$ absorbing random walks of length at most l. We count p_n, the number of endpoints that are labeled $+$ and q_n, the number of endpoints that are labeled $-$. Then we use $\widetilde{p_s} = \frac{p_n}{\hat{N}}$ to estimate p_s and use $\widetilde{q_s} = \frac{q_n}{\hat{N}}$ to estimate q_s. In addition, let β denote the fraction of random walks that are forced to stop at length l. In other words, β is the fraction of random walks that might end up at voters if there is no maximum length restriction.

If we denote p_s^* and q_s^* to be the estimates of p_s and q_s respectively using the absorbing random walks without the maximum length restriction, then by Lemma 21, w.h.p. p_s^* is bounded by

$$p_s - \sqrt{p_s\epsilon} - \epsilon \leq p_s^* \leq p_s + \sqrt{p_s\epsilon} + \epsilon, \tag{6.9}$$

and w.h.p. q_s^* is also bounded by

$$q_s - \sqrt{q_s\epsilon} - \epsilon \leq q_s^* \leq q_s + \sqrt{q_s\epsilon} + \epsilon. \tag{6.10}$$

Since $\widetilde{p_s} \leq p_s^* \leq \widetilde{p_s} + \beta$, $p_s - \sqrt{p_s\epsilon} - \epsilon - \beta \leq \widetilde{p_s} \leq p_s + \sqrt{p_s\epsilon} + \epsilon$ w.h.p. Similarly, $q_s - \sqrt{q_s\epsilon} - \epsilon - \beta \leq \widetilde{q_s} \leq q_s + \sqrt{q_s\epsilon} + \epsilon$ w.h.p. Let $\widetilde{r_s} = \widetilde{p_s} - \widetilde{q_s}$, then $r_s - \sqrt{p_s\epsilon} - \sqrt{q_s\epsilon} - 2\epsilon - \beta \leq \widetilde{r_s} \leq r_s + \sqrt{p_s\epsilon} + \sqrt{q_s\epsilon} + 2\epsilon + \beta$ w.h.p. Since $\sqrt{xy} \leq \frac{x+y}{2}$ for any non-negative real numbers x and y, $q_s - \sqrt{q_s\epsilon} - \epsilon - \beta \geq \frac{q_s}{2} - \frac{3}{2}\epsilon - \beta$. Then we get $q_s \leq 2(\widetilde{q_s} + \frac{3}{2}\epsilon + \beta)$, and similarly $p_s \leq 2(\widetilde{p_s} + \frac{3}{2}\epsilon + \beta)$. With the help of previous inequalities, we can bound $\widetilde{r_s}$ using the inequality

$$|r_s - \widetilde{r_s}| \leq \sqrt{2\left(\widetilde{p_s} + \frac{3}{2}\epsilon + \beta\right)\epsilon} + \sqrt{2\left(\widetilde{q_s} + \frac{3}{2}\epsilon + \beta\right)\epsilon} + 2\epsilon + \beta. \tag{6.11}$$

In order to make $\widetilde{r_s}$ a good approximation for r_s, both ϵ and β should be very small. We can increase the number of absorbing random walks to decrease ϵ. We can also increase the maximum length l to decrease β.

Chapter 7

Conclusion and Future Work

In this thesis, we have considered the problem of efficiently sampling the endpoints of random walks in the edge sampling model. The model assumes that there is an oracle that uniformly samples an edge from a graph per query. Inspired by the SingleRandomWalk algorithm suggested in paper [42] for the graph streaming model, first of all, we designed the Sampling-AS algorithm which is an Atlantic City randomized algorithm. It samples the endpoint of a single random walk with given length l, unless some unlikely "bad" event R happens. Sampling-AS works in two phases. In Phase 1, the algorithm prepares a large number of w-length random walks in parallel. In Phase 2, the algorithm starts a random walk L_u and uses the prepared w-length random walks to extend L_u so that the length of L_u reaches l quickly. Next, we developed a Las Vegas algorithm called Sampling-LS that does not have "bad" events and always samples the endpoint of a single random walk; however, its performance is not as good as the performance of Sampling-AS.

Then we considered the performance of both algorithms on special classes of graphs such as regular graphs, random graphs and fast mixing graphs. We find that both algorithms achieve better performance after some modifications. Finally, based on Sampling-AS, we suggested the Sampling-AM algorithm which samples the endpoints of multiple random walks, unless some unlikely "bad" event R_M happens. It is then used to estimate PageRank scores for vertices, and it is also used in random walk recommendation systems.

Some suggestions for future work include:

- One possible direction is generalizing the edge sampling model to weighted graphs. The oracle on weighted graph G can be assumed to return an edge e of weight w_e with probability $\frac{w_e}{w_G}$ (where $w_G = 2\sum_{e\in E} w_e$). The edge sampling model we have studied in this thesis is only a special case where all edges are equally weighted. The problem becomes how to sample the endpoints of weighted random walks. We have started some initial work to address this problem and the direction is promising.

- Another possible direction is finding and proving a probabilistic bound on the number of rounds required by the Sampling-LS algorithm on directed graphs. Lemma 12

is important for bounding the number of rounds required by Sampling-LS on undirected graphs. However, we proved that Lemma 12 does not hold on directed graphs. Therefore, another approach is required to bound the number of rounds needed when preparing w-length random walks on directed graphs.

Bibliography

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[2] Reid Andersen, Christian Borgs, Jennifer Chayes, Uriel Feige, Abraham Flaxman, Adam Kalai, Vahab Mirrokni, and Moshe Tennenholtz. Trust-based recommendation systems: an axiomatic approach. In *Proceedings of the 17th international conference on World Wide Web*, pages 199–208. ACM, 2008.

[3] László Babai. Monte-carlo algorithms in graph isomorphism testing. *Université tde Montréal Technical Report, DMS*, (79-10), 1979.

[4] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

[5] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[6] Surender Baswana. Streaming algorithm for graph spanners-single pass and constant processing time per edge. *Information Processing Letters*, 106(3):110–114, 2008.

[7] Dave Bayer and Persi Diaconis. Trailing the dovetail shuffle to its lair. *The Annals of Applied Probability*, pages 294–313, 1992.

[8] Itai Benjamini, Gady Kozma, and Nicholas Wormald. The mixing time of the giant component of a random graph. *arXiv preprint math/0610459*, 2006.

[9] Béla Bollobás. *Random graphs*. Springer, 1998.

[10] Béla Bollobás. The evolution of random graphs - the giant component. In *Random Graphs*, 2001.

[11] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

[12] Andrei Broder. Generating random spanning trees. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 442–447. IEEE, 1989.

[13] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 253–262. ACM, 2006.

[14] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[15] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *Journal of the ACM (JACM)*, 60(1):2, 2013.

[16] Derek J de Solla Price. Networks of scientific papers. *Science*, 149(3683):510–515, 1965.

[17] Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. In *Automata, Languages and Programming*, pages 716–727. Springer, 2007.

[18] Paul Erdös and Alfréd Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[19] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

[20] Nikolaos Fountoulakis and Bruce Reed. The evolution of the mixing rate. *arXiv preprint math/0701474*, 2007.

[21] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[22] Sudipto Guha. Tight results for clustering and summarizing data streams. In *Proceedings of the 12th International Conference on Database Theory*, pages 268–275. ACM, 2009.

[23] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM journal on computing*, 18(6):1149–1178, 1989.

[24] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.

[25] David R Karger. Using randomized sparsification to approximate minimum cuts. In *SODA*, volume 94, pages 424–432, 1994.

[26] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[27] David R Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43. ACM, 2004.

[28] David Kempe and Jon Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 471–480. IEEE, 2002.

[29] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM, 2000.

[30] Ching Law and Kai-Yeung Siu. Distributed construction of random expander networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 2133–2143. IEEE, 2003.

[31] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

[32] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[33] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 165–178. Springer, 2008.

[34] Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

[35] Nicholas Metopolis, Arianna Rosenbluth, Augusta Teller, and Edward Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[36] Milena Mihail, Christos Papadimitriou, and Amin Saberi. On certain connectivity properties of the internet topology. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 28–35. IEEE, 2003.

[37] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[38] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. 1999.

[39] Aaron Plavnick. The fundamental theorem of Markov chains. *University of Chicago VIGRE REU*, 2008.

[40] Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 390–403. ACM, 2010.

[41] Pili Hu Ruohan Gao, Huanle Xu and N Wing Cheong Lau. Accelerating graph mining algorithms via uniform random edge sampling. *IEEE ICC*, 2016.

[42] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating PageRank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011.

[43] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed PageRank computation. In *Distributed Computing and Networking*, pages 11–26. Springer, 2013.

[44] William J Turner. Black box linear algebra with the linbox library. 2002.

[45] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.