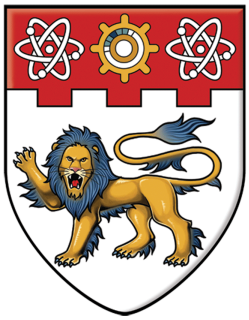
RANDOM WALKS IN DISTRIBUTED NETWORKS AND THEIR APPLICATIONS



ANISUR RAHAMAN MOLLA

Division of Mathematical Sciences

School of Physical and Mathematical Sciences

A thesis submitted to the Nanyang Technological University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

2014

Acknowledgments

First and foremost, with great pleasure, I express my heartfelt gratitude to my supervisor Professor Gopal Pandurangan for his guidance and support throughout my time in NTU. His amazing way of explanation during the analysis of algorithms always makes me feel excited to be involved with the problem. Also I want to mention his excellent teaching, especially his course on randomized algorithms, which helped a lot. Above all and the most important, he provided me unflinching encouragement and support in various ways. I will forever be indebted to him for providing me with the scope to work under his supervision and for his invaluable guidance.

I am extremely grateful to my collaborators. Especially, a large fraction of my gratitude goes to my coauthor Atish Das Sarma for providing me a wonderful research experience from the first stage. It is fun to work with Atish. I am also thankful to John Augustine for his help and encouragement. I would like to thank Peter Robinson and Danupon Nanongkai, whom I could find beside me whenever I needed them.

I would like to extend my gratitude to Ehab Morsy. His presence and rational attitude makes life more easy to tackle. I also sincerely thank all my lab mates, fellow graduate students and friends for providing me a great research and live experience in NTU. I thank to all my well wishers who helped me directly or indirectly throughout my student life.

I am thankful to Professor Rana Barua of Indian Statistical Institute, Kolkata and Professor Avishek Adhikari of Department of Pure Mathematics, Calcutta University. Prof. Rana Barua was my master (M.Tech.) dissertation advisor and helped me significantly during my time at ISI. Avishek da is helping me from my M.Sc. level, whenever I asked for anything.

Finally and most importantly, I take this opportunity to express my love and respect for my family, who always makes me feel that I am under a safe roof in this world. I can not thank them enough, I just want to mention their love,

scarifies and tolerance during my study. My parents, elder brother, sister and my brother in law are always having a tremendous influence on my life. My father in law and mother in law also deserves special acknowledgements. They are my true well wisher and I got another family in them. My wife Sabnam, is my great source of inspiration and encouragement. She has given me an unbelievable amount of support, love, and happiness during my graduate studies. *This thesis is dedicated to them.*

Abstract

This thesis studies random walks and its algorithmic applications in distributed networks. Random walk is a fundamental technique which has found numerous applications in computer science, mathematics, statistics, physics, and among others. The simulation of random walks in a network is an important tool, with a lot of applications in algorithms and complexity theory. In particular, in communication networks, random walks have been used in various applications including token management, load balancing, network topology discovery and construction, search, and peer-to-peer membership management, local and lightweight algorithms for dynamic networks etc. While several such algorithms are ubiquitous, and use random walk sampling, the walks themselves have always been performed in a naive way — simply passing the random walk token from one node to its neighbor. Thus, to perform a random walk of length ℓ, the naive approach requires ℓ steps. Therefore, a natural question is whether a better algorithm is possible in the distributed model. In this thesis, we focus on two main questions: (1) How efficiently random walks can be performed in distributed networks? (2) How random walks can be used in designing efficient distributed algorithms for important distributed computing problems?

Towards the first question, the thesis studies efficient distributed random walk sampling in networks, where we focus on both static and dynamic networks. For static networks, we present a round and message optimal algorithm which can be used to output several random walk samples in a continuous online fashion. This significantly improves upon both the naive technique that requires linear time and messages, and the sophisticated algorithm presented by Das Sarma et al. [42] which has the same sub-linear (quadratic) running time as our algorithm, but requires a large number of messages (depending on the number of edges in the network). Moreover, we perform a comprehensive experimental evaluation on numerous network topologies which proves the effectiveness and efficiency of our algorithm. For dynamic networks, we present a fast distributed algorithm

for performing random walks. Our algorithm is the first-known algorithm that provably speeds up random walks in dynamic networks. Furthermore, we show a key application of this random walk algorithm for the fundamental problem of information spreading (a.k.a *gossip*) in dynamic networks. We use our random walk algorithm to obtain a fast distributed information spreading algorithm.

Towards the second question, we study two important algorithmic applications: (1) Distributed computation of PageRank (2) Distributed computation of sparse cuts. PageRank has emerged as a powerful measure of relative importance of nodes in a network. In distributed computing, PageRank vectors have been used for several different applications ranging from determining important nodes, load balancing, search, and identifying connectivity structures. We devise random walk-based algorithms for computing PageRank and prove strong bounds on the round complexity. Our algorithms are the first and fully distributed algorithms for computing PageRank with provably efficient running time.

Finding sparse cuts is an important tool in analyzing large-scale distributed networks such as the Internet and Peer-to-Peer networks, as well as large-scale graphs such as the web graph, online social communities, and VLSI circuits. Sparse cuts are particularly useful in graph clustering and partitioning. We develop fast distributed algorithms for computing sparse cuts in distributed networks, where random walks are used as a key ingredient.

Contents

Acknowledgments i
Abstract iii
List of Figures ix

1 Introduction **1**
1.1 Preliminaries 6
1.2 Distributed Computing Model 7
1.3 Contributions of This Thesis: Problems, Results and Road Map . 9

2 Random Walk Sampling in Distributed Networks **15**
2.1 Introduction 16
2.1.1 Network Model 16
2.1.2 Overview of Our Results 17
2.1.3 Applications and Previous Work 18
2.1.4 Outline of This Chapter 19
2.2 Theoretical Analysis of Algorithms 20
2.2.1 Algorithm descriptions 20
2.2.2 Previous Results - Rounds and Messages 21
2.2.3 Round Complexity 24

2.2.4 Message Complexity 25
2.3 Concentration Bounds on κ 26
2.3.1 Extensions to different walk lengths 28
2.4 Experiments 29
2.4.1 Short walk utilization factor κ 32
2.4.2 Message complexity plots 35
2.5 Conclusion 38

3 Random Walks in Dynamic Networks and Applications 39
3.1 Introduction 40
3.1.1 Outline of This Chapter 40
3.1.2 Network Model and Definitions 41
Dynamic Networks 41
Model of Computation 42
3.2 Random Walks in a Dynamic Graph 43
3.2.1 Mixing Time of a Dynamic Graph 44
3.2.2 Monotonicity property of the distribution vector 48
3.3 Overview of Our Results 48
3.4 Related Work and Technical Overview 51
3.5 Overview of Subsequent Work 54
3.6 Algorithm for Single Random Walk 55
3.6.1 Description of the Algorithm 55
3.6.2 Analysis 60
Correctness 60
Time Analysis 62
3.6.3 Generalization to Non-regular Dynamic Graphs 74
3.7 Algorithm for κ Random Walks 74
3.8 Application: Information Dissemination (or k-Gossip) 78
3.8.1 Analysis 79
3.9 Conclusion 83

4 Distributed PageRank Computation 85

4.1 Introduction . . . 86

4.1.1 Outline of This Chapter . . . 86

4.1.2 Overview of Our Results . . . 87

4.1.3 The Model of Computation . . . 88

4.1.4 PageRank Background . . . 88

4.2 A Distributed Algorithm for PageRank . . . 89

4.2.1 Analysis . . . 92

4.2.2 Correctness of PageRank Approximation . . . 92

4.2.3 Time Complexity . . . 94

4.3 A Faster Distributed PageRank Algorithm (for Undirected Graphs) 95

4.3.1 Description of Our Algorithm . . . 95

4.3.2 Analysis . . . 100

4.4 Conclusion . . . 103

5 Distributed Sparse Cuts Computation 105

5.1 Introduction . . . 106

5.1.1 Model, Notations and Definitions . . . 107

5.1.2 Overview of Our Results . . . 108

5.1.3 Outline of This Chapter . . . 110

5.2 A Distributed Algorithm for Sparse Cut . . . 110

5.2.1 Estimating Random Walk Probability Distribution . . . 111

5.2.2 Computation of Sparse Cut . . . 114

5.2.3 Description and Analysis of the Algorithm . . . 116

5.2.4 Time Complexity Analysis . . . 119

5.3 Finding Local Cluster Set . . . 121

5.4 Sparse Cuts using PageRank . . . 122

5.4.1 Estimating Personalized PageRank . . . 123

Analysis . . . 124

5.4.2 Algorithm for Sparse Cut using PageRank . . . 126

5.5 Lower Bound 128
5.6 Conclusion 129

6 Conclusion and Further Study **131**

List of Publications **134**

Bibliography **134**

List of Figures

2.1 Varying length of the walk ℓ. $n = 10K, \eta = 1, \lambda = \sqrt{\ell}$ 32
2.2 Varying number of nodes n. $\ell = n$, $\eta = 1, \lambda = \sqrt{\ell}$ 33
2.3 Varying number of short walks η. $n = 10K, \ell = n, \lambda = \sqrt{\ell}$ 33
2.4 Varying length of short walk λ. $n = 10K, \ell = n, \eta = 1$ 34
2.5 Varying length of the walk ℓ. $n = 10K, \eta = 1, \lambda = \sqrt{\ell}$ 35
2.6 Varying number of nodes n. ℓ =n, $\eta = 1, \lambda = \sqrt{\ell}$ 36
2.7 Varying number of short walks η. $n = 10K, \ell = n, \lambda = \sqrt{\ell}$ 37
2.8 Varying length of short walk λ. $n = 10K, \ell = n, \eta = 1$ 37

3.1 Figure illustrating the Algorithm of stitching short walks together. (Figure is taken from [42]). 57

5.1 Node j computes the number of its neighbors that are in the left side and right side of j in the ordered vertex set π. 116

Chapter 1

Introduction

Random walk in graphs is a well known paradigm, which plays a crucial role in computer science, including distributed computing. The simulation of random walks in a graph is a fundamental technique which has a high impact in algorithms and complexity theory. Algorithms in many applications use random walks as an integral subroutine. In distributed computing alone, it has found remarkably wide range of applications, in both theory and practice. For instance, applications of random walks in communication networks include network topology construction [50, 68, 70], constructing random spanning trees [11, 14, 22], load balancing [61], token management [18, 32, 58], small-world routing [64], querying in wireless ad-hoc networks and sensor networks [88], group communication in ad-hoc network [44], search [1, 30, 50, 51, 73, 93], information spreading and gathering over a network [2], load balancing [61], computing PageRank on the web [57], distributed construction of expander networks [68], checking expander [45], monitoring overlays [78], peer-to-peer membership management [48, 94], and several others. They are particularly useful in providing uniform and efficient solutions to distributed control of dynamic networks [23].

One of the most effective and commonly used application of random walks in networks is node sampling (from some distribution)[1]. Naturally, the efficiency of

[1]Random walks can be used to sample from arbitrary distribution by changing the hop

the algorithms, which use random walks as a key subroutine, depend on how fast one can perform random walks in networks. While the sampling requirements may differ in different applications, whenever a true sample is required from a random walk (of certain length), typically all applications perform the walk in a naive way. The naive algorithm is simply sending the random walk token from one node to its neighbor (randomly) for ℓ steps to compute a walk of length ℓ. Thus, the naive algorithm takes time and messages that is linear with respect to ℓ, to compute a single random walk of length ℓ. Such an algorithm may not scale well when the network size becomes larger and hence one may ask a natural question: can we perform such sampling with a significantly less number of rounds (and messages) in distributed networks?

Das Sarma et al. [40, 41, 42] were the first to solve this problem by a distributed algorithm whose running time is significantly faster than the naive ℓ bound. They proposed an approach to perform a single random walk of length ℓ in $\tilde{O}(\sqrt{\ell D})$ rounds (where D is network's diameter). This is the first algorithm with sub-linear (in the length of the walk) running time to perform random walks in distributed networks. The high-level idea used in the $\tilde{O}(\sqrt{\ell D})$-round distributed algorithm is to "prepare" a few short walks in the beginning (executed in parallel) and then carefully stitch these walks together later as necessary. More precisely, the algorithm works in two phases: in first phase, each node performs many short-length random walks in parallel. This is done in naive way by forwarding (many) tokens containing the ID of nodes. Then in the second phase, the short walks (created in first phase) are stitched to compute the desired walk of certain length. Later, it was shown in [82] that the above distributed algorithm is optimal in round complexity for performing a single random walk. In this thesis, we develop distributed random walk algorithms that build on this approach.

While the above algorithm in [42] is optimal in round complexity and scalable i.e., it works under CONGEST model of distributed computing (see Section 1.2

probabilities, e.g., see Metropolis-Hastings sampling [42, 56].

for precise description of CONGEST model), however, a main drawback of this result is that it incurred a message complexity of $\Omega(m\sqrt{\ell})$ (m is the number of edges). Hence, their algorithm requires a large number of messages for every random walk, depending on the number of edges in the network. Moreover, they only considered performing a single walk, or a few walks, which yields a few samples. Most applications, however, require several random walk samples in a continuous manner. In this thesis, we first consider this continuous random walk sampling problem and present round and message optimal algorithm which computes several random walk samples in a continuous online fashion.

The sub-linear time random walk algorithm of Das Sarma et al. [42] applied only to *static* networks. A major problem left open in [42] is whether a similar approach can be used to speed up random walks in *dynamic* networks. In this thesis, we extend the study of distributed random walks to dynamic networks. Modern communication networks are subject to changes of their topology over time. Examples include real world networks such as peer-to-peer networks, internet, social networks, wireless ad-hoc networks, sensor networks etc. Dynamic graphs try to capture this changing behavior of real-world networks. In dynamic networks, where the topology can change arbitrarily from round to round, extensive distributed algorithmic techniques that have been developed in the last few decades for *static* networks (see e.g., [74, 86, 90]) are not readily applicable. On the other hand, we would like distributed algorithms to work correctly and terminate even in networks that keep changing continuously over time (not assuming any eventual stabilization). Random walks being so simple and very local (i.e., each subsequent step in the walk depends only on the neighbors of the current node and does not depend on the topological changes taking place elsewhere in the network) can serve as a powerful tool to design distributed algorithms for such highly dynamic networks. However, it remains a challenge to show that one can indeed use random walks to solve non-trivial distributed computation problems efficiently in such networks, with provable guarantees. This thesis is a

step towards this direction. In particular, we first develop a rigorous framework for studying random walks in a dynamic network. We present a fast distributed random walk algorithm that runs in sub-linear time on the length of the walk, to perform a single random walk and returns a node sample that is "close" to the stationary distribution of the dynamic network. Our algorithm is the first-known algorithm that provably speeds up random walks in dynamic networks. We also extend our algorithm to efficiently perform and return multiple independent random walk samples. We further show a key application of our fast random walk sampling algorithm in dynamic networks. We present a fast distributed algorithm for the fundamental problem of *information dissemination* in a dynamic network.

This thesis also focuses on how random walks can be applied as a key algorithmic tool to solve classical graph problems in distributed computing. We present random walk based distributed algorithms for two important problems — *PageRank* computation and *Sparse Cuts* computation.

PageRank has gained importance in a wide range of applications and domains, ever since it first proved to be effective in determining node importance in large graphs. In fact, it was a pioneering idea behind Google's search engine. Computing PageRank and its variants efficiently in various computation models has been of tremendous research interest in both academia and industry. In distributed computing alone, PageRank vectors have been used for several different applications ranging from determining important nodes, load balancing, search, and identifying connectivity structures. Surprisingly, however, there has been little work towards designing provably efficient fully-distributed algorithms for computing PageRank. This is perhaps because the traditional method of computing PageRank vectors is to apply iterative methods i.e., do matrix-vector multiplications till (near)-convergence. Since such techniques may not adapt well in certain settings, when dealing with a global network with only local views as is common in distributed networks such as Peer-to-Peer (P2P) networks, and particularly, very large networks, it becomes crucial to design far more efficient

techniques. Therefore, PageRank computation using Monte Carlo methods (i.e., random walk style method) is more appropriate in a distributed model where only limited sized messages are allowed through each edge in each round. We take all these concerns into consideration and design efficient and fully decentralized algorithms for computing PageRank vectors in distributed networks.

Finding sparse cuts is an important tool in analyzing large-scale distributed networks such as the Internet and peer-to-peer networks, as well as large-scale graphs such as the Web graph, online social communities, and VLSI circuits. Sparse cuts are useful in graph clustering and partitioning among numerous other applications. In distributed communication networks, they are useful for topology maintenance and for designing better search and routing algorithms. Sparse cuts are those cuts that have low conductance and can be used to determine well-connected clusters[2] and thus also identify potential "bottlenecks" in the network. In particular, the edges crossing the cut can be considered as *critical* edges and they have been used in designing algorithms to improve searching, topology maintenance (i.e., maintaining a well-connected topology), and reducing routing congestion in networks [49]. Such algorithms are useful in the design, analysis, and maintenance of *topology aware* networks [49].

Since there are exponential number of cuts in the network, it is significantly more challenging to efficiently find the sparsest cut or approximate it in a distributed fashion. Hence computing sparse cuts needs a different approach compared to computing conductances and mixing time as in the works of [42, 62]. In this thesis, we develop fast distributed algorithms for computing sparse cuts in networks.

The rest of this chapter is organized as follows. Section 1.1 introduces some basic definitions and notations that will be used throughout the thesis. Section 1.2 discusses the model of distributed computation. Section 1.3 gives a roadmap

[2]A cut $(S, V \setminus S)$ is a partition of the set of nodes V into S (assume $|S| \leq |V|/2$) and $V \setminus S$. A low conductance cut has lot more edges within S than those going outside S, and hence S is relatively well (intra)connected.

of the rest of this thesis, where we formally state the problems and our results.

1.1 Preliminaries

We provide some basic definitions and concepts which will be used repeatedly throughout this thesis.

Graphs are denoted by $G = (V, E)$, where V is the set of vertices and E is the set of edges. We use the terminology graph and network; node and vertex; edge and (communication) link interchangeably throughout the thesis. Furthermore, we assume G is an *undirected* graph (unless otherwise stated). As usual, n is the number of vertices $|V|$ and m is the number of edges $|E|$. The degree of a vertex $v \in V$ is denoted by $\deg(v)$ or in short by $d(v)$. A graph is called d-regular if the degree of all the nodes is d.

The *Simple Random walk* in a graph G is a stochastic process in which the walk starts from a source node and in each step, the walk moves from the current node to a random neighbor, i.e., from the current node v, the probability to move in the next step to a neighbor u is $\Pr(v, u) = 1/d(v)$ for $(v, u) \in E$ and 0 otherwise. This is also called as *standard random walk*.

Suppose a random walk starts from a node v_0 and after t steps it reached node v_t. Then we get a probability distribution P_t on v_t; the initial distribution starts with probability 1 at node v_0. We say that the distribution P_r is *stationary* (or steady-state) if $P_{t+1} = P_t$ for all $t \geq r$. Let π denote the stationary distribution vector. It is known that for every (undirected) graph G, the distribution $\pi(v) = d(v)/2m$ is stationary. In particular, for a regular graph the stationary distribution is the uniform distribution. The *mixing time* of a random walk in a graph G is the maximum time taken to reach "close" to the stationary distribution of the graph, starting from any node.

Throughout this thesis, we often use the $\tilde{O}$ notation which hides a polylogarithmic factor in the number of nodes in the network, i.e., $\tilde{O}(f)$ equals a quantity

$O(f \log^c n)$, for some constant c. Furthermore, we will often use the term "with high probability" or in short "w.h.p", which means with probability at least $1 - 1/n^{\Omega(1)}$, where n is the number of nodes in the network.

1.2 Distributed Computing Model

We model the communication network as an unweighted, connected n-node graph $G = (V, E)$. Unless specified otherwise, we restrict our attention to *undirected* and *unweighted* graphs throughout the thesis. Initially, every node has limited knowledge about the network. Nodes do not have any information about the graph topology or any other global properties of the graph at the beginning of the computation. Specifically, we assume that each node is associated with a distinct identity number (e.g., its IP address). At the beginning of the computation, each node v accepts as input its own identity number (which is of length $O(\log n)$ bits) and the identity numbers of its neighbors in G. The node may also accept some additional inputs as specified by the problem at hand (e.g., the number of nodes, edges or diameter of the network). The nodes are allowed to communicate through the edges of the graph G. We assume that the communication occurs in synchronous *rounds*, i.e., nodes run at the same processing speed and any message that is sent by some node v to its neighbors in some round r will be received by the end of round r. In particular, all the nodes wake up simultaneously at the beginning of round 1, and from this point on the nodes always know the number of the current round. To ensure scalability, we restrict the number of bits that are processed and sent per round by each node to be polylogarithmic in n, the network size. In particular, in each round, each node is allowed to send a message of size B bits (where B is polylogarithmic in n) through each communication link. This is a widely used standard model, called the CONGEST(B) model to study distributed algorithms (e.g., see [84, 86]) and captures the bandwidth constraints inherent in real-world computer networks. Typically, as assumed

here, $B = O(\log n)$, which is number of bits needed to send a node id in a n-node network, however in some chapter we may assume B to be polylogarithmic in n. We note that there is another distributed computing model where this bandwidth restriction is relaxed. This is called the LOCAL model [86], which allows a polynomial (in n) number of bits to be sent across a link per round.

Sometime we assume additional communication between nodes such as— a node can communicate with any node if it knows the other node's address (e.g., its IP address). In particular, our algorithm presented in Section 4.3 of Chapter 4 assumes this direct communication.

There are several measures of efficiency (e.g., number of rounds, messages etc.) of distributed algorithms, but in most of the chapters, we will focus on *the running time*, i.e. the number of *rounds* of distributed communication. Note that the computation that is performed by the nodes locally is "free", i.e., it does not affect the number of rounds; however, we will only perform polynomial cost computation locally in any node. We note that in the CONGEST model, it is rather trivial to solve a problem in $O(m)$ rounds, where m is the number of edges in the network, since the entire topology (all the edges) can be collected at one node and the problem solved locally. The goal is to design faster algorithms.

While this is a nice theoretical abstraction, it still does not motivate the most natural practical difficulties. A well established concern with this model is that for simple operations, the entire network may spawn a large number of parallel messages in order to minimize rounds. This can be very expensive from a practical standpoint. A critical component in the analysis of practical algorithms is the overall message complexity per execution of any algorithm. This becomes even more crucial from the standpoint of continuous processing of algorithms, perhaps even in parallel. Therefore, the goal is to design algorithms that have a low amortized message complexity and minimize the worst case round complexity, both simultaneously. Due to their conflicting nature, few algorithms perform well on both metrics. We note that we focus on both the round and message

complexity of our algorithm presented in Chapter 2.

1.3 Contributions of This Thesis: Problems, Results and Road Map

Here, we formally state the problems considered in this thesis, overview of our results, and structure of this thesis.

[Chapter 2] ***The Random Walk Sampling Problem.*** We consider the problem of performing random walk sampling efficiently in a distributed network. We are given an arbitrary undirected, unweighted, and connected n–node network $G = (V, E)$ and a random walk request length ℓ. The goal is to devise a distributed algorithm such that, the algorithm continuously accepts source node inputs s, and after each input, the algorithm performs a random walk of length ℓ and outputs the ID of a node v which is randomly picked according to the probability that it is the destination of a random walk of length ℓ starting at s. Our goal is to output a true random sample from the ℓ-walk distribution starting from s. Once the sample has been output, a new request is issued to the algorithm, and this proceeds in a continual manner. Every sample node is independent of the previous one. The objective is to minimize the round complexity as well as the message complexity for each of these requests.

In particular, we present an algorithm which shows how several random walks can be performed continuously, such that each walk of length ℓ can be performed exactly in just $\tilde{O}(\sqrt{\ell D})$ rounds (where D is the diameter of the network), and $O(\ell)$ messages. Our algorithm uses the same general approach as [42] but exploits certain key observation to achieve the significant improvement in message complexity over the algorithm of [42]. The algorithm in [42] prepared 'many' short random walks initially and later stitched those short walks to get the required long random walk. However, most of the short walks were unused when they compute

a single random walk. Our algorithm crucially uses these unused short walks to compute multiple random walks. In particular, we show a technical result that if the source nodes are chosen randomly proportional to the node degrees, our algorithm uses up a constant fraction of these short walks. This improves our algorithm significantly in message complexity. Moreover, we present extensive experimental evaluations which further show that our techniques perform very well in various network topologies. This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan [37].

[Chapter 3] ***Performing Random Walks in Dynamic Networks and Applications.*** We investigate efficient distributed computation in *dynamic* networks in which the network topology changes arbitrarily from round to round. We develop a framework for the design and analysis of distributed random walk algorithms in dynamic networks with applications. In particular, the goals of this chapter are two fold: (1) giving fast distributed algorithms for performing random walk sampling efficiently in dynamic networks, and (2) applying random walks as a key subroutine to solve non-trivial distributed computation problems in dynamic networks. Towards the first goal, we first present a rigorous framework for studying random walks in a dynamic network. This is necessary, since it is not immediately obvious what the output of random walk sampling in a changing network means. The main purpose of our random walk algorithm is to output a random sample close to the "stationary distribution" of the underlying dynamic network. Our random walk algorithms work under an oblivious adversary that fully controls the dynamic network topology, but does not know the random choices made by the algorithms. We present a fast distributed random walk algorithm that runs in $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ rounds with high probability, where $\mathcal{T}$ is (an upper bound on) the *dynamic mixing time* and $\mathcal{D}$ is the *dynamic diameter* of the network respectively. The dynamic diameter is the maximum time taken to broadcast an information on the dynamic network and it is bounded above by the number of nodes in the network. Our algorithm uses small-sized messages

only and returns a node sample that is "close" to the stationary distribution of the dynamic network. That is, our algorithm runs in $\tilde{O}(\sqrt{\mathcal{TD}})$ rounds to perform a random walk of length $\mathcal{T}$ in dynamic networks. We further extend our algorithm to efficiently perform and return κ independent random walk samples in $\tilde{O}(\min\{\sqrt{\kappa\mathcal{TD}}, \kappa + \mathcal{T}\})$ rounds. This is directly useful in the application considered in this chapter.

Towards the second goal, we present a key application of our fast random walk sampling algorithm. We present a fast distributed algorithm for the fundamental problem of *information dissemination* (also called as *gossip*) in a dynamic network. In gossip, or more generally, k-gossip, there are k pieces of information (or tokens) that are initially present in some nodes and the problem is to disseminate the k tokens to all nodes. In an n-node network, solving n-gossip allows nodes to distributively compute any computable function of their initial inputs using messages of size $O(\log n + d)$, where d is the size of the input to the single node [65]. We present a random-walk based algorithm that runs in $\tilde{O}(\min\{n^{1/3}k^{2/3}(\mathcal{TD})^{1/3}, k\mathcal{D}\})$ rounds with high probability. This gives the first $o(k\mathcal{D})$-time fully-distributed *token forwarding* algorithm that improves over the previous-best $O(k\mathcal{D})$ round distributed algorithm [65], albeit under an oblivious adversarial model. A lower bound of $\Omega(nk/\log n)$ under the adaptive adversarial model of [65], was shown in [46]; hence one cannot do substantially better than the $O(nk)$ algorithm in general under an adaptive adversary. This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan [36].

We like to mention that the random walk framework developed in this thesis for dynamic networks, has also subsequently proved useful in developing efficient storage and search algorithms as well as developing fast byzantine agreement algorithms in dynamic networks (cf. Section 3.5).

[Chapter 4] ***Distributed PageRank Computation Problem.*** We consider the problem of computing PageRank in graphs. PageRank is essentially the

stationary distribution of a slightly modified standard random walk process. The power and applicability of PageRank arises from its basic intuition of being a way to naturally identify 'important' nodes, or in certain cases, similarity between nodes. We present random walk-based distributed algorithms for computing PageRank in general graphs and prove strong bounds on the round complexity. In particular, we first present a distributed algorithm that takes $O(\log n/\epsilon)$ rounds with high probability on any graph (directed or undirected), where n is the network size and ϵ is the reset probability used in the PageRank computation (typically ϵ is a fixed constant). We then present a faster algorithm that takes $O(\sqrt{\log n}/\epsilon)$ rounds in undirected graphs. Both of the above algorithms are scalable, as each node sends only small ($\operatorname{polylog} n$) number of bits over each edge per round. These are the first fully distributed algorithms for computing PageRank vectors with provably efficient running time. This chapter is based on joint work with Atish Das Sarma, Gopal Pandurangan and Eli Upfal [39].

[Chapter 5] ***Distributed Sparse Cut Computation Problem.*** Then, we focus on the problem of sparse cuts computation in distributed networks. Given that $G = (V, E)$ be an undirected graph with conductance ϕ, the problem is to find a cut set $(S, V \setminus S)$, where $S \subseteq V$ such that the conductance of $(S, V \setminus S)$ is close to ϕ. We present two distributed algorithms that output a cut of conductance at most $\tilde{O}(\sqrt{\phi})$ with high probability, in $\tilde{O}(\frac{1}{b}(\frac{1}{\phi} + n))$ rounds, where b is balance of the cut of given conductance. In particular, to find a cut of constant balance (i.e., the cuts are of approximately equal size), our algorithm takes $\tilde{O}(\frac{1}{\phi}+n)$ rounds and finds such a cut (if it exists) with similar approximation. The second algorithm is a variant of the first one and builds on the PageRank-based approach. Our algorithms can also be used to output a *local* cluster, i.e., a subset of vertices near a given source node, and whose conductance is within a quadratic factor of the best possible cluster around the specified node. Both our distributed algorithms can work without knowledge of the optimal ϕ value and hence can be

used to find approximate conductance values both globally and locally with respect to a given source node. On a high-level, our approach is based on efficiently implementing the algorithms from Lovász-Simonovits [71], Spielman-Teng [89], and Andersen et al. [4] in the CONGEST distributed computing model. However, there are several challenging issues to overcome in distributed model. In particular, the above running time bounds follow from a technical contribution on computing conductances of n different cuts in linear time. We further give a lower bound on the time needed for any distributed algorithm to compute any non-trivial sparse cut. We show that any distributed approximation algorithm for computing the sparsest cut will take $\tilde{\Omega}(\sqrt{n}+D)$ rounds, where D is the diameter of the graph. This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan [38].

[Chapter 6] We summarize the main contributions of this thesis and discuss some interesting problems for further study.

Chapter 2

Random Walk Sampling in Distributed Networks

In this chapter, we consider the problem of performing random walk sampling efficiently in a distributed network[1]. Recall that in this problem, we are given an arbitrary undirected, unweighted, and connected graph G and a random walk request length ℓ. The goal is to devise a distributed algorithm such that, the algorithm continuously accepts source node inputs s, and after each input, the algorithm performs a random walk of length ℓ and outputs the ID of a node v which is randomly picked according to the probability that it is the destination of a random walk of length ℓ starting at s. We assume the standard random walk throughout this chapter. We present both round and message optimal distributed algorithms that present a significant improvement on all previous approaches. The theoretical analysis and comprehensive experimental evaluation of our algorithms show that they perform very well in different types of networks of differing topologies.

In particular, our results show how several random walks can be performed continuously (when source nodes are provided only at runtime, i.e., online), such

[1]This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan and contains material from [37].

that each walk of length ℓ can be performed exactly in just $\tilde{O}(\sqrt{\ell D})$ rounds (where D is the diameter of the network), and $O(\ell)$ messages. This significantly improves upon both, the naive technique that requires $O(\ell)$ rounds and $O(\ell)$ messages, and the sophisticated algorithm of [42] that has the same round complexity as our but requires $\Omega(m\sqrt{\ell})$ messages (where m is the number of edges in the network). Our theoretical results are corroborated through extensive experiments on various topological data sets.

2.1 Introduction

The topic of this chapter is node sampling through performing random walks in networks. The important algorithmic applications of random walks has make them attractive in communication networks. Though algorithms in different applications use random walks as an integral subroutine in different way, a key purpose of random walks in network applications is to perform node sampling. Random walk-based sampling is simple, local, and robust. Random walks also require little index or state maintenance which make them especially attractive to self-organizing dynamic networks such as Internet overlay and ad hoc wireless networks [23, 93]. In this chapter, we present efficient distributed random walk sampling algorithm in networks that are significantly faster than the existing and naive approaches and at the same time achieve optimal message complexity. Further, our experimental results show that our techniques perform very well in various network topologies.

2.1.1 Network Model

We assume the communication network as an undirected, unweighted, connected graph $G = (V, E)$ with $|V| = n, |E| = m$. We consider the standard CONGEST model of distributed computation. The detailed description of this model is given in Section 1.2. A well established concern with this model is that for simple

operations, the entire network may spawn a large number of parallel messages in order to minimize rounds. This can be very expensive from a practical standpoint. A critical component in the analysis of practical algorithms is the overall message complexity per execution of any algorithm. This becomes even more crucial from the standpoint of continuous processing of algorithms, perhaps even in parallel. Therefore, the goal is to design algorithms that have a low amortized message complexity and minimize the worst case round complexity, both simultaneously. Due to their conflicting nature, few algorithms perform well on both metrics. In this chapter we present an algorithm that is near-optimal in terms of messages as well as rounds in parallel.

2.1.2 Overview of Our Results

- We introduce the problem of continuous processing of random walks. The objective is for a network to support a continuous sequence of random walk requests from various source nodes and perform node sampling to minimize round and message complexity for each request.

- We present the first algorithm that is efficient in both round complexity and message complexity. Our technique and analysis presents almost-tight bounds on the message and round complexity in a widely used network congestion model.

- We perform comprehensive experimental evaluation on numerous topological networks and highlight the effectiveness and efficiency of our algorithm. The experimental results corroborate the theoretical contributions and show that our random walk sampling algorithm performs very well on various metrics for all parameter ranges.

2.1.3 Applications and Previous Work

Random walks have been used in a wide variety of applications in distributed networks as mentioned previously. We describe here some of the applications in more detail.

Morales and Gupta [78] discuss about discovering a consistent and available monitoring overlay for a distributed system. For each node, one needs to select and discover a list of nodes that would monitor it. The monitoring set of nodes need to satisfy some structural properties such as consistency, verifiability, load balancing, and randomness, among others. This is where random walks come in. Random walks are a natural way to discover a set of random nodes that are spread out (and hence scalable), that can in turn be used to monitor their local neighborhoods. Random walks have been used for this purpose in another paper by Ganesh et al. [48] on peer-to-peer membership management for gossip-based protocols. Morales and Gupta [79, 80] have several more papers in their line of work on AVMON system and similar systems that use several continuous node samples as a way to monitor distributed systems.

Speeding up distributed algorithms using random walks has been considered for a long time. Besides our approach of speeding up the random walk itself, one popular approach is to reduce the *cover time*. Recently, Alon et. al. [3] show that performing several random walks in parallel reduces the cover time in various types of graphs. They assert that the problem with performing random walks is often the latency. In these scenarios where many walks are performed, our results could help avoid too much latency and yield an additional speed-up factor.

A nice application of random walks is in the design and analysis of expanders. We mention two results here. Law and Siu [68] consider the problem of constructing expander graphs in a distributed fashion. One of the key subroutines in their algorithm is to perform several random walks from the specified source nodes. Dolev and Tzachar [45] use random walks to check if a given graph is an expander. The first algorithm given in [45] is essentially to run a random

walk of length $n \log n$ and mark every visited vertex. Later, it is checked if all vertices have been visited. Broder [22] and Wilson [92] gave algorithms to generate random spanning trees using random walks and Broder's algorithm was later applied to the network setting by Bar-Ilan and Zernik [14]. Recently Goyal et al. [52] show how to construct an expander/sparsifier using random spanning trees. The approach of Broder and Wilson's algorithm is essentially performing a random walk from a specified node (root node) until all nodes are visited. For each non-root node, output the edge that is used for its first visit. The running time of their algorithm is bounded by the time to visit all the nodes of the graph (i.e., the cover time) which could be $\tilde{O}(mD)$ in the worst case. This can be easily implemented in the distributed setting by performing random walk in a naive way, which has the same running time of $\tilde{O}(mD)$. In [42], Das Sarma et al. implemented the Broder's algorithm in distributed setting in an arbitrary graph using their optimal random walk algorithm. (We note that the work of Bar-Ilan and Zernik constructed random spanning trees only in rings and complete graphs.) The improved random spanning tree algorithm can be useful in implementing Goyal et al. [52]'s approach in a distributed way. A variety of such applications can greatly benefit from the random walk sampling techniques presented in this chapter.

2.1.4 Outline of This Chapter

In the next section (cf. Section 2.2), we present our algorithms, and message and round complexity analyses. This rests on some concentration analysis of key random walk properties of our algorithm that we then prove in Section 2.3. Finally, we present extensive experiments on various topological networks in Section 2.4.

2.2 Theoretical Analysis of Algorithms

2.2.1 Algorithm descriptions

We first describe the algorithm for single random walk in [42] and then describe how to extend this idea for continuous random walks. The current algorithm is also randomized and we focus more on the message complexity. The high-level idea for single random walk is to perform many short random walks in parallel and later stitch them together [42]. Then for multiple random walks we choose the source node randomly each time and perform single random walk using the same set of short length walks.

Our main algorithm for performing continuous random walk each of length ℓ is described in Continuous-Random-Walk (cf. Algorithm 3). This algorithm uses other algorithms Pre-Processing (cf. Algorithm 1) and Single-Random-Walk (cf. Algorithm 2). The Pre-Processing function is called only one time at the beginning of Continuous-Random-Walk, to perform $\eta d(v) \log n$ short walks of length λ from each vertex v; once these pre-processed short walks are insufficient to answer a single random walk request, only then is the pre-processing table reconstructed and the algorithm resumes answering single random walk requests accessing the short length walks from the new table. At the end of Pre-Processing, each vertex knows the destination IDs of the short walks that it initiated. We note that, since all short random walks are independent to each other, we can retain the unused short walks from the previous pre-processing table. This does not give any improvement to the asymptotic bounds, however, this slight optimization could be useful in practical scenario.

Algorithm 1 PRE-PROCESSING(η, λ)

Input: number of short walks of each node v is $\eta deg(v) \log n$, and desired short walk lengths λ.

Output: set of short random walks of each nodes

Each node v performs $\eta_v = \eta \deg(v) \log n$ random walks of length $\lambda + r_i$ where r_i (for each $1 \leq i \leq \eta$) is chosen independently at random in the range $[0, \lambda - 1]$.

1: Let $r_{max} = \max_{1 \leq i \leq \eta} r_i$, the random numbers chosen independently for each of the η_x walks.
2: Each node x constructs η_x messages containing its ID and in addition, the i-th message contains the desired walk length of $\lambda + r_i$.
3: **for** $i = 1$ to $\lambda + r_{max}$ **do**
4: Each node v does the following: Consider each message M held by v and received in the $(i-1)$-th iteration (having current counter $i - 1$). If the message M's desired walk length is at most i, then v stored the ID of the source (v is the desired destination). Else, v picks a neighbor u uniformly at random and forward M to u after incrementing its counter.
5: **end for**
6: Send the destination IDs back to the respective sources (this can be done by sending the destination IDs along the "reverse" path).

2.2.2 Previous Results - Rounds and Messages

We first restate the main round complexity theory for SINGLE-RANDOM-WALK and also state the message complexity of this algorithm.

Lemma 2.1 (Theorem 3.6 in [42]). *For any ℓ, Algorithm* SINGLE-RANDOM-WALK *([42]) solves the Single Random Walk Problem and, with probability at least $1 - \frac{2}{n}$, finishes in $\Theta\left(\lambda \eta \log n + \frac{\ell D}{\lambda}\right)$ rounds.*

Algorithm 2 SINGLE-RANDOM-WALK(s, ℓ)

Input: Starting node s, and desired walk length ℓ.

Output: Destination node of the walk outputs the ID of s.

Stitch $\Theta(\ell/\lambda)$ walks, each of length in $[\lambda, 2\lambda - 1]$

1: The source node s creates a message called "token" which contains the ID of s
2: The algorithm generates a set of *connectors*, denoted by C, as follows. (Connectors are the endpoints of the short walks, i.e., the points where we stitch.)
3: Initialize $C = \{s\}$
4: **while** Length of walk completed is at most $\ell - 2\lambda$ **do**
5: Let v be the node that is currently holding the token.
6: v uniformly chooses one of its short length sample and let v' be the sampled value if any exists (which is a destination of an unused random walk of length between λ and $2\lambda - 1$).
7: **if** $v' =$ NULL (all walks from v have already been used up) **then**
8: Algorithm terminates failing this walk.
9: **end if**
10: v sends the token to v'
11: $C = C \cup \{v\}$
12: **end while**
13: Walk naively until ℓ steps are completed (this is at most another 2λ steps)
14: A node holding the token outputs the ID of s

Lemma 2.2. *The message complexity of* SINGLE-RANDOM-WALK *algorithm is* $O\left(\eta\lambda m \log n + \frac{\ell D}{\lambda}\right)$ *where m is number of edges and D is the diameter of the network.*

Proof. For computing $\eta deg(v) \log n$ short walks of length λ it uses $\Theta(\lambda \eta deg(v) \log n)$

messages. Since for a single short walk of length λ it sends λ messages and hence for n nodes it requires $\Theta(\lambda\eta \log n \sum_v deg(v)) = \Theta(\lambda\eta m \log n)$ messages. For stitching one short walk with another we need to contact the destination ID. This can be done quickly by using a BFS tree. Note that the BFS tree needs to be constructed only once [2] ($\Theta(m)$ messages) and each stitch uses $O(D)$ messages. Combining these, the lemma follows. □

Algorithm 3 CONTINUOUS-RANDOM-WALK(ℓ)

Input: ℓ.

Output: Continuous ℓ length random walk samples from sources nodes presented adversarially or randomly.

Source nodes S**:** The source node of each walk of length ℓ can be presented adversarially or randomly accordingly to some distribution. Let this continuous sequence of source nodes be denoted by ordered set S.

1: Call PRE-PROCESSING($\eta = 1,\ \lambda = 24\sqrt{\ell D}(\log n)^3$)
2: **while** Indefinitely **do**
3: **while** Algorithm does not fail (algorithm gets stuck due to insufficient short walks) **do**
4: Select the next source node s from the ordered set S.
5: call SINGLE-RANDOM-WALK(s, ℓ).
6: If Single-Random-Walk returns with fail, exit loop
7: **end while**
8: Call PRE-PROCESSING($\eta = 1,\ \lambda = 24\sqrt{\ell D}(\log n)^3$) again and use this table.
9: Rerun request for s and then continue subsequent walks based on random samples.
10: **end while**

[2] If we assume that nodes have access to shortest path routing table, then BFS tree is not needed.

In networks such as P2P or overlay networks, if we assume that a node can access quickly (in constant time) another node whose ID (IP address) is known, then one can improve the time and message complexity of stitching, saving a $\Theta(D)$ factor.

We now analyze the round and message complexity for CONTINUOUS-RANDOM-WALK algorithm in the next two subsections. For simplified analysis, we use κ to denote the fraction of short walks of the pre-processing table that get used, before the algorithm fails and needs to rerun the pre-processing stage. The next two subsections assume a value of κ and prove bounds using it. In the following section, we actually present bounds on κ itself to arrive at the main theorem of this chapter. To recall other notation, $\eta_v = \eta deg(v) \log n$ is the number of short length walks pre-processed for each node v, λ is the length of these short walks, n is the number of nodes, m is the number of edges, and D is the diameter of the network.

2.2.3 Round Complexity

Lemma 2.3. *For any ℓ, Algorithm* CONTINUOUS-RANDOM-WALK *(cf. Algorithm 3) serves continuous random walk requests such that, with probability at least $1 - \frac{2}{n}$, the total number of rounds used until* PRE-PROCESSING *needs to be invoked for a second time is $O(\lambda\eta \log n + \kappa m\eta D \log n)$, where κ is the fraction of used short length walks from the preprocessing table.*

Proof. The proof is same as Theorem 3.6 in [42] for Single Random Walk; the only difference is we are doing continuous walks of same length ℓ. Therefore for Continuous Walks, if κ is the fraction of used short length walks form the preprocessing table, then a total $O(\kappa m\eta \log n)$ short walks are used. Hence we need to stitch $O(\kappa m\eta \log n)$ times. Clearly, each stitching can take at most $O(D)$ rounds (cf. Lemma 3.11 in [42]). Therefore, it contributes $O(\kappa m\eta \log nD)$ rounds. Hence total $O(\lambda\eta \log n + \kappa m\eta D \log n)$ rounds, since pre-processing table construction takes $O(\lambda\eta \log n)$ rounds. □

Corollary 2.1. *The average number of rounds per random walk of length ℓ of* Continuous-Random-Walk *(cf. Algorithm 3) is $O\left(\frac{\ell}{\kappa}(\frac{1}{m}+\frac{\kappa D}{\lambda})\right)$ with high probability.*

Proof. The total number of random walks of length ℓ that have been completed successfully by Continuous-Random-Walk is $\Theta\left(\frac{\kappa m\eta\lambda\log n}{\ell}\right)$, as total $O(\kappa m\eta\log n)$ short walks each of length λ have been used. Hence the bound on the average number of rounds per walk follows. □

2.2.4 Message Complexity

Lemma 2.4. *The message complexity of* Continuous-Random-Walk *algorithm is $O\left(\eta\lambda m\log n+\kappa m\eta D\log n\right)$, until* Pre-Processing *needs to be invoked for a second time, where κ is fraction of used short length walks from the preprocessing table.*

Proof. The message complexity of the stage of Pre-Processing is as before. Further, for each subsequent ℓ length walk request, an additional $O(D\ell/\lambda)$ messages are used. Also, as before we know that the total number of random walks of length ℓ that have been completed successfully by Continuous-Random-Walk is $\Theta\left(\frac{\kappa m\eta\lambda\log n}{\ell}\right)$, as total $O(\kappa m\eta\log n)$ short walks each of length λ have been used. Therefore the contribution from this towards the total message complexity is $O(D\ell/\lambda * \frac{\kappa m\eta\lambda\log n}{\ell})$ which reduces to $O(mD\eta\kappa\log n)$. Combining these, the lemma follows. □

Corollary 2.2. *The average number of messages per random walk of length ℓ of* Continuous-Random-Walk *is $O\left(\frac{\ell}{\kappa}(1+\frac{\kappa D}{\lambda})\right)$.*

Proof. From the above Lemma 2.4 we know that the total number of messages used for computing all walks of Continuous-Random-Walk algorithm is $O\left(\eta\lambda m\log n+\kappa m\eta D\log n\right)$. Now the total number of walks of length ℓ is $O\left(\frac{\kappa m\eta\lambda\log n}{\ell}\right)$, as total $O(\kappa m\eta\log n)$ short walks each of length λ. Hence we get the average number of messages per walk by dividing by this. □

Combining the above two corollaries, we get the following.

Lemma 2.5. *The average number of rounds and messages per random walk of length ℓ of* Continuous-Random-Walk *(cf. Algorithm 3) are $O\left(\frac{\ell}{\kappa}(\frac{\log n}{m} + \frac{\kappa D}{\lambda})\right)$ and $O\left(\frac{\ell}{\kappa}(1 + \frac{\kappa D}{\lambda})\right)$ respectively.*

Corollary 2.3. *For our choice of $\lambda = \tilde{\Theta}(\sqrt{\ell D})$, the average rounds and messages per random walk becomes $\tilde{O}\left(\frac{\ell}{\kappa m} + \sqrt{\ell D} + D\right)$ and $O\left(\frac{\ell}{\kappa} + D\right)$ respectively.*

Proof. If we put $\lambda = \tilde{\Theta}(\sqrt{\ell D})$ in Lemma 2.5 then the average round and message becomes $\tilde{O}\left(\frac{\ell}{\kappa m} + \sqrt{\ell D}\right)$ and $O\left(\frac{\ell}{\kappa} + \sqrt{\ell D}\right)$ respectively. Now $\sqrt{\ell D} \leq \ell + D$. So the corollary follows. □

Note that, in the above corollary, κ (< 1) can be small, so that the bounds can become large. We show in the next section that, under certain condition, κ is a constant and hence our bounds are almost optimal.

2.3 Concentration Bounds on κ

The goal of this section is to present a lower bound on κ, the fraction of rows of the pre-processing table (or the fraction of all short walks) that get used before the algorithm fails to perform a random walk request, and needs to rerun the pre-processing stage. All the analysis in this section assumes that sources S in Continuous-Random-Walk are sampled according to the degree distribution. While the algorithm Continuous-Random-Walk remains meaningful otherwise also, our proofs crucially rely on this random sampling of sources. Obtaining similar theorems for more general sequence of sources in S remains an open question.

We now present the central theorem that lower bounds κ.

Theorem 2.1. *Given any graph G, if* Continuous-Random-Walk *is invoked on $\ell = O(m)$ and the source nodes S are chosen randomly proportional to the*

node degrees, then the algorithm uses up at least $\kappa = \Omega(1)$ fraction of all short walks in Pre-Processing *table, before a request fails and a second call needs to be made to* Pre-Processing.

Proof. Assume for now that we do $d(v)$ short walks for each vertex v. The total number of walks of length ℓ is $T = \frac{2m\lambda}{\ell}$ if all the short walks are used. Let $K = \alpha T$, where α is a constant in $[0, 1]$. Note that if we manage to perform K walks of length ℓ, then we have utilized a constant fraction of the short walks. For one ℓ-length walk, in expectation a vertex v can be a connector at most $\frac{d(v)\ell}{2m\lambda}$ times (by linearity of expectation). (Connectors are the endpoints of the short walks, i.e., the points where we stitch. Note that only when a vertex is visited as a connector we end up using a short walk initiated from that vertex.) Then for K walks, each of length ℓ, the expected number of times that v is visited as a connector vertex is $K\frac{d(v)\ell}{2m\lambda} = \alpha d(v)$. Let N denote the number of times the vertex v is visited as a connector in K walks. By above, $E[N] = \alpha d(v)$. By Markov's inequality, $\Pr(N \geq d(v)) \leq \frac{\alpha d(v)}{d(v)} = \alpha$. Now consider the above experiment (for a fixed vertex v) repeated $c \log n$ independent times for some constant c, that is suitably large. (In other words, assume that we do $cd(v) \log n$ short walks — total over all experiments — from each node v.) We say that an experiment is "success" if $N < d(v)$. If we have success, then that means that we have done K walks of length ℓ (and hence utilized a constant fraction of the $d(v)$ short walks) for that experiment before a request fails. By above, the probability of success is at least some constant $\alpha' = 1 - \alpha$. Let $X_1^v, X_2^v, \ldots, X_{c \log n}^v$ be the 0-1 indicator random variables such that $X_i^v = 1$ (if success occurs in i-th time) and zero otherwise. Let $X^v = \sum_{1=1}^{c \log n} X_i^v$. Then $E[X^v] = \alpha' c \log n$. Since the variables are independent, by Chernoff's bound $\Pr(|X^v - E[X^v]| \geq c' \log n) \leq e^{-\frac{2c'^2 \log^2 n}{c \log n}} \leq \frac{1}{n^2}$, for a suitable constant $c \leq (c')^2$. Therefore, $\Pr(|X^v - E[X^v]| \geq c' \log n) \leq \frac{1}{n^2}$. Thus, at least a constant fraction of the experiments succeed with probability $1 - 1/n^2$. By union bound [77], the total number of visits to every vertex v as connector in all ($c \log n$ times K) walks is at most $O(\alpha d(v) c \log n)$ with probability at least $1 - 1/n$. This

implies that the total number of short walks utilized is a constant fraction of the best possible, before a request fails. □

We now present the main theorem of this chapter, which follows from the above bound on κ stated in Theorem 2.1, and the message and round complexity bounds in terms of κ stated in Corollary 2.3. Notice that this presents optimal round and message complexities simultaneously for every walk (since independently also $\Omega(\ell + D)$ is a clear lower bound on the number of messages for a single ℓ-length random walk, and $\Omega(\sqrt{\ell D} + D)$ is a nontrivial lower bound on the number of rounds for a single ℓ-length random walk as shown in [82]).

Theorem 2.2. *Algorithm* Continuous-Random-Walks *satisfies walk requests continuously and indefinitely such that the amortized message complexity per walk is $O(\ell + D)$ and with high probability, every single walk request completes in $\tilde{O}(\sqrt{\ell D} + D)$ rounds, provided the source nodes are chosen randomly proportional to the node degrees in the graph.*

2.3.1 Extensions to different walk lengths

While our main algorithm of Continuous-Random-Walk and the associated theorems are stated for a fixed ℓ, they can be generalized to handle different walk lengths. We omit the rigorous details for brevity and present a brief explanation of the generalization here. The theorems and experiments go through verbatim for this case as well.

Suppose that Continuous-Random-Walk is designed to not only support new source node requests each time but also new length requests for the random walks. One can of course store multiple Pre-Processing tables, one for each associated ℓ_i, in the entire allowed range for ℓ. This way, when a request is presented, the appropriate Pre-Processing table is accessed and the corresponding short walks queried. Then, whenever Continuous-Random-Walk fails on a specific single random walk request, only this Pre-Processing is

rerun, and answering the random walk requests resume.

While this does solve the problem and guarantees the identical throughput and efficiency, a practical concern is that performing and storing so many short walks, corresponding to multiple different lengths, can be expensive. There is a simple way to counter this, by storing short walks in a doubling fashion. In particular, if each ℓ_i was in the range $[1, n]$, instead of storing short walks corresponding to each of $\ell_i = 1, 2, 3, \ldots, n-1, n$, we perform short walks only corresponding to $\ell_i = 1, 2, 4, \ldots, n/2, n$. This exponentially reduces the number of short walks at each node, or the number of pre-processing tables, from n to $\log n$. Now, whenever a walk request for ℓ_i is received, it can be answered by just performing a longer walk, of length $\tilde{\ell}_i$ such that $\ell_i \leq \tilde{\ell}_i \leq 2\ell_i$.

2.4 Experiments

We have used the following five important graph generative models for experiments. Several of these have been used in other papers as well for random walk experiments, see for e.g. [51]. These graphs together cover a nice spectrum of fast mixing to slow mixing, uniform degrees to very skewed degrees, small diameter to large diameter, etc. thereby testing the algorithm in all the extreme cases as well as nice cases.

- Regular Expander: We worked on the most commonly studied random graph model of $G(n, p)$. Here, each of the $n(n-1)/2$ edges occurs independently and randomly with probability p. We choose p as $\log n/n$ so that the expected number of edges is roughly $(n \log n)/2$. Further, the expected degree of every vertex is $\log n$. This, with high probability, results in a graph with good expansion and it is regular graph in expectation i.e., the expected degree is same.

- Two-tier topologies with clustering: First we construct four isolated roughly regular expanders, as mentioned above in $G(n, p)$, of the same size - think

of these as independent clusters. Then from each cluster we pick a small number of nodes (roughly one-fourth the size of the cluster and connect them using another $G(n,p)$ - think of this as a tier-two cluster. Again we use the same value of p as above.

- Power-law graphs: In distributed settings, many important networks are known to have power-laws. We use the well known preferential attachment growth model to construct random power-law graphs. The essential process proceeds by starting with a small clique (of same 5 nodes), and then adding vertices sequentially. Each subsequent vertex added connects with an edge to each of the previous edges with probability depending on their degrees, and independently. Specifically, the new vertex connects with a previous vertex v with probability proportional to $deg(v)^\alpha$ where the exponent α is a parameter.

- Random Geometric Graph: A random geometric graph is a random undirected graph drawn on a bounded region $[0,1) \times [0,1)$. It is generated by placing n vertices uniformly at random and independently on the region (i.e. both the x and y coordinates are picked uniformly and independently). Then edges are constructed deterministically - two vertices u and v are connected by an edge if and only if the distance between them is at most a parameter threshold r. We choose r as $\sqrt{\frac{\log n}{n}}$ so that the degree of each vertices is $O(\log n)$ w.h.p.

- Grid Graph: Consider a square grid graph ($\sqrt{n} \times \sqrt{n}$) which is a Cartesian product of two path graphs with $\sqrt{n}$ vertices on each. Since a path graph is a median graph, the square grid graph is also a median graph. All grid graphs are bipartite (since they have no odd length cycles).

We compute and maintain a preprocessing table containing $\eta_v = \eta deg(v) \log n$ short walks of length λ from each vertex v. We then check how many walks of length ℓ can be done using this table before we hit a node all of whose short

walks have been exhausted. The source nodes for each of the ℓ length random walk requests are sampled randomly according to the degree distribution.

We perform experiments on each of the aforementioned synthetically generated graphs, and also by varying different parameters. In particular, we conduct separate experiments for each of (a) varying the length of the walk (ℓ) as a function of n, (b) varying the number of nodes(n), (c) varying the length of the short walk (λ) stored by the preprocessing table, and (d) varying the number of short walks stored from each node as a function of the parameter η. For each of these, we use certain default values when a specific parameter is being varied for a plot, while others are held constant. The default values we use are $n = 10{,}000$, $\ell = n$, $\eta = 1$, and $\lambda = \sqrt{\ell}$.

Since we are interested in how many random walks of length ℓ can be done in a continuous manner with small round and message complexity, this translates to analyzing the utilization for one specific pre-processing table before Continuous-Random-Walk gets stuck and needs to invoke another call to Pre-Processing. In particular, to analyze the round complexity, we conduct a set of experiments to evaluate κ, the number of rows of the Pre-Processing table used before the algorithm fails (κ plotted on the y-axis). As mentioned in the previous section, this gives a bound of ℓ/κ on the round complexity. In particular, if κ is a constant, and large enough, this shows excellent utilization and an asymptotically optimal round complexity. Similarly, for message complexity, we explicitly conduct a second set of plots that calculates the message complexity on the y-axis based on κ and D, for easier visualization.

We plot graphs by varying each of the parameters ℓ, n, λ, η. Each figure contains fives lines, one for each of the above network models: For each of these plot values, we perform ten different runs and then present the average value.

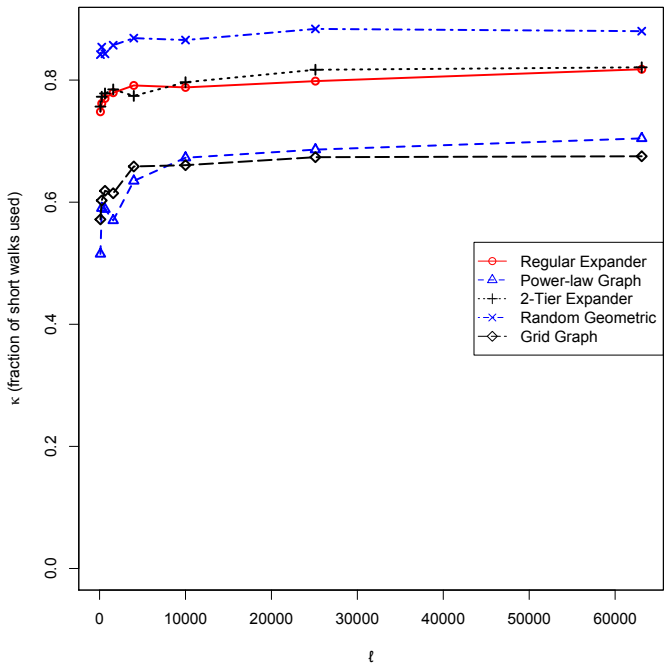


Figure 2.1: Varying length of the walk ℓ. $n = 10K, \eta = 1, \lambda = \sqrt{\ell}$

2.4.1 Short walk utilization factor κ

Varying ℓ [Figure 2.1]: Here n is fixed at 10,000 and ℓ is varying as $n^{0.5}, n^{0.6}, ..., n^{1.2}$; λ is $\sqrt{\ell}$ and η is 1. In this case we see that at least 50% of the pre-processed short walk rows are used up. This utilization is even better for some of the graph topologies such as $G(n, p)$ and two-tier clustering graph and reaches around 80%. Therefore, for the entire range of ℓ being small to very large, our algorithm performs extremely well: In particular, κ is a large constant and therefore the round complexity and message complexity are close to optimal - i.e. within a constant factor of the best possible.

Varying n [Figure 2.2]: The number of nodes n is varying between 1000 and 10,000. We see that in all of the graphs, the utilization of pre-processed short walks before the algorithm terminates is at least 60%. We also see that in some of the graphs, the utilization is substantially higher. There even as the graph size scales, our performance remains equally good.

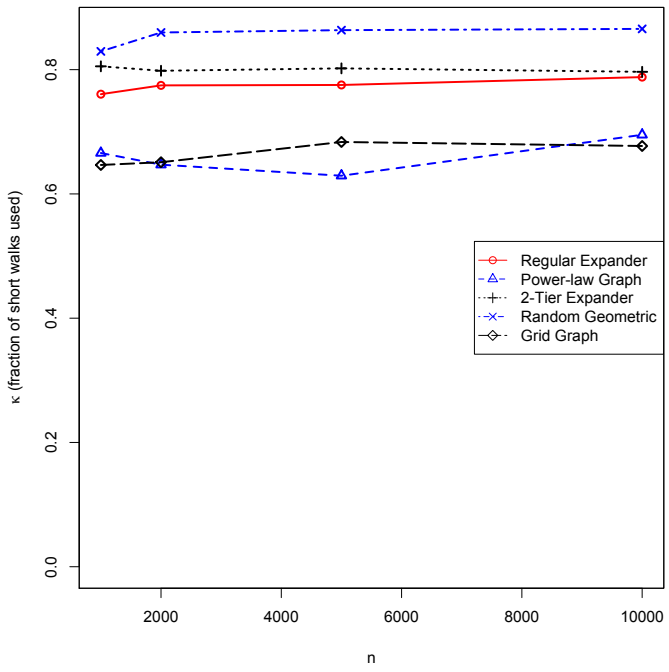


Figure 2.2: Varying number of nodes n. $\ell = n$, $\eta = 1, \lambda = \sqrt{\ell}$

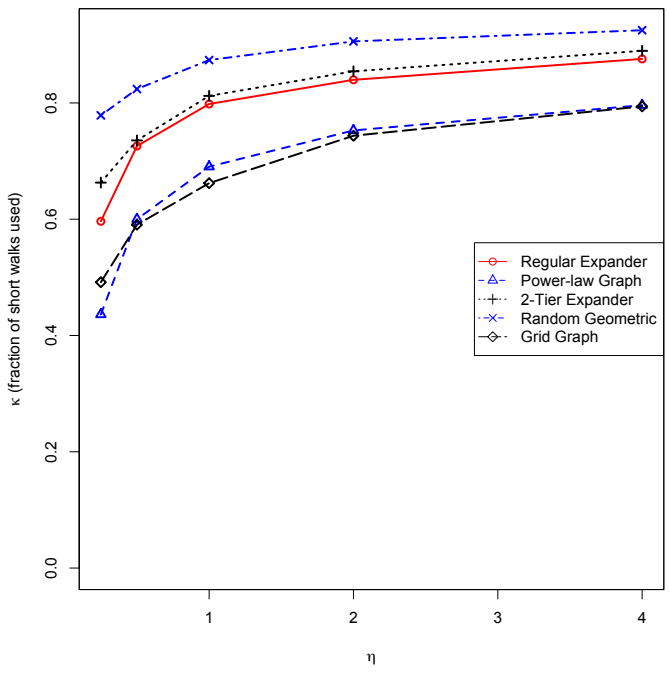


Figure 2.3: Varying number of short walks η. $n = 10K, \ell = n, \lambda = \sqrt{\ell}$

Varying η [Figure 2.3]: We see that the used fraction of rows is increasing with the number of short length walk η. We see that even for small enough η of 1, on all experiments, the utilization κ on the y-axis is at least 0.6, or 60% of all the short walks get used. This means that for each node v, the number of short length walk $d(v) \log n$ suffice, therefore the round and message complexity remain near-optimal as proved previously.

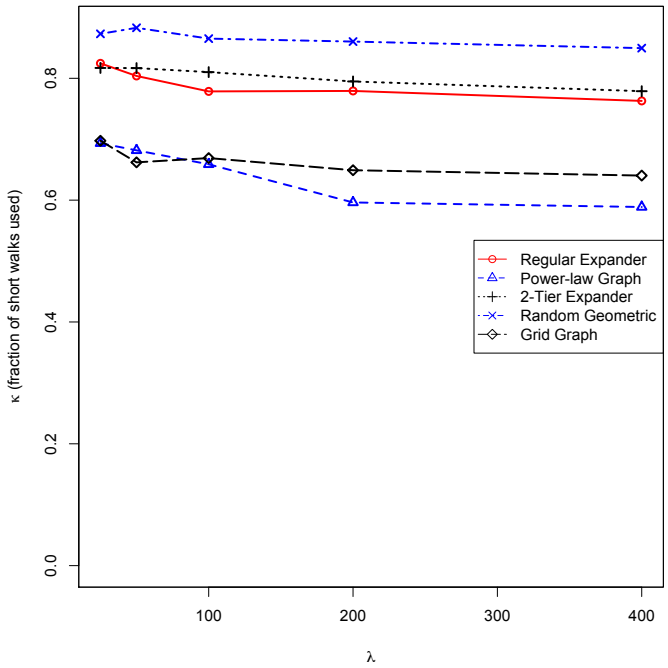


Figure 2.4: Varying length of short walk λ. $n = 10K, \ell = n, \eta = 1$

Varying λ [Figure 2.4]: The default value of λ is $\sqrt{\ell}$. In this plot, we vary λ from $0.25\sqrt{\ell}$ to $\sqrt{\ell}$ in doubling steps. We see that the utilization roughly remains the same throughout the plot. Even though the algorithm needs to choose λ to optimize for rounds and messages, this plot shows that for any of the values, it performs well.

Summary of observed round complexity: To summarize the plots for varying different parameters on the x-axis, we see that in all the plots, the value of κ on the y-axis is a constant and usually at least 0.5. Since κ is 1 for optimal or perfect utilization of the table, we see that for all parameter values, the utilization

is only a small constant factor (around 2) away from the optimal. Therefore, the round complexity, as proven in the previous section, increases only marginally.

2.4.2 Message complexity plots

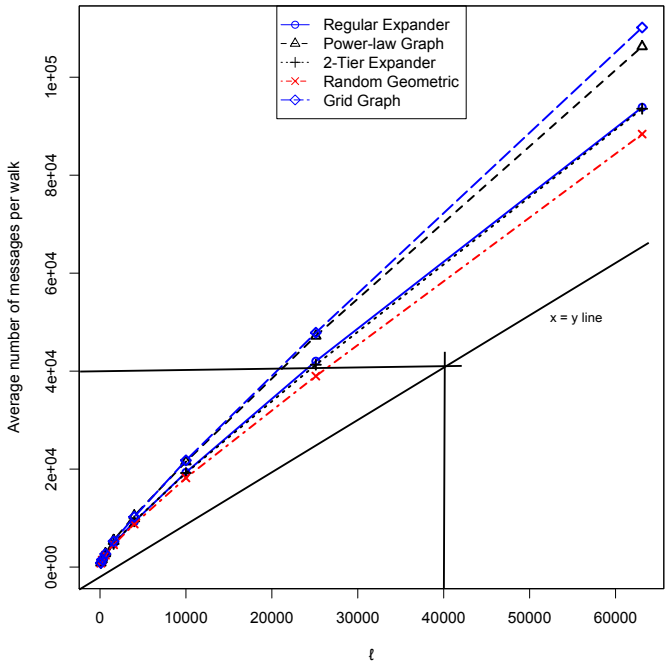


Figure 2.5: Varying length of the walk ℓ. $n = 10K, \eta = 1, \lambda = \sqrt{\ell}$

Varying ℓ [Figure 2.5]: In this plot, we vary ℓ and note the message complexity of the algorithm Continuous-Random-Walk, per random walk Single-Random-Walk request within it. For any walk ℓ the optimal number of messages would be ℓ itself. Notice that in our plot also, all the lines (that is for all the graphs) are very close to the $x = y$ line, which is the optimal line. Therefore, the efficiency of Continuous-Random-Walk amortized is almost the best possible.

Varying n [Figure 2.6]: In this plot as well, since we use the default value of $\ell = n$, the best possibility is for the message complexity to be n, which corresponds to the $x = y$ line. Notice that again for all the graphs, the lines

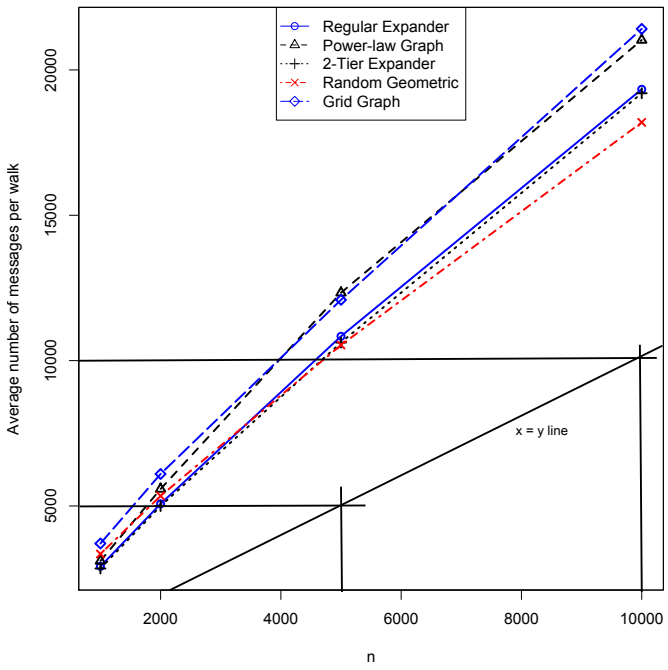


Figure 2.6: Varying number of nodes n. ℓ =n, $\eta = 1, \lambda = \sqrt{\ell}$

for message complexity, through the entire range, is almost the best possible; this is because we get straight lines with the slope being very close to $x = y$ line.

Varying η [Figure 2.7]: As η is increased between 0.25 and 4, we see that the message complexity reduces rapidly. It is expected that as the number of pre-processing rows are increased, the efficiency would improve and therefore message complexity also improves. This plots sharp decline, however, also suggests that just a small enough η is also sufficient to drastically bring down the message complexity close to optimal, regardless of what the graph topology is.

Varying λ [Figure 2.8]: This plot is very similar to that of varying η, here we see again that as λ is increased, the message complexity goes down rapidly. Recall that here we are comparing different λ values for a fixed ℓ value of n. Our algorithm Continuous-Random-Walk uses $\lambda = \sqrt{\ell}$ but we tried this plot with even smaller values of λ. As expected, the message complexity is high initially, however, as λ is increased close to $\sqrt{\ell}$, the message complexity rapidly reduces, and improves the algorithm performance substantially.

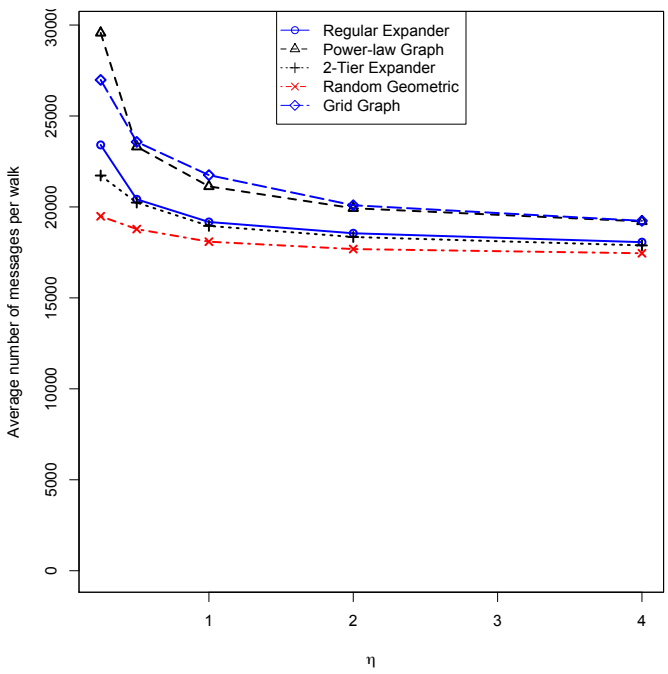


Figure 2.7: Varying number of short walks η. $n = 10K, \ell = n, \lambda = \sqrt{\ell}$

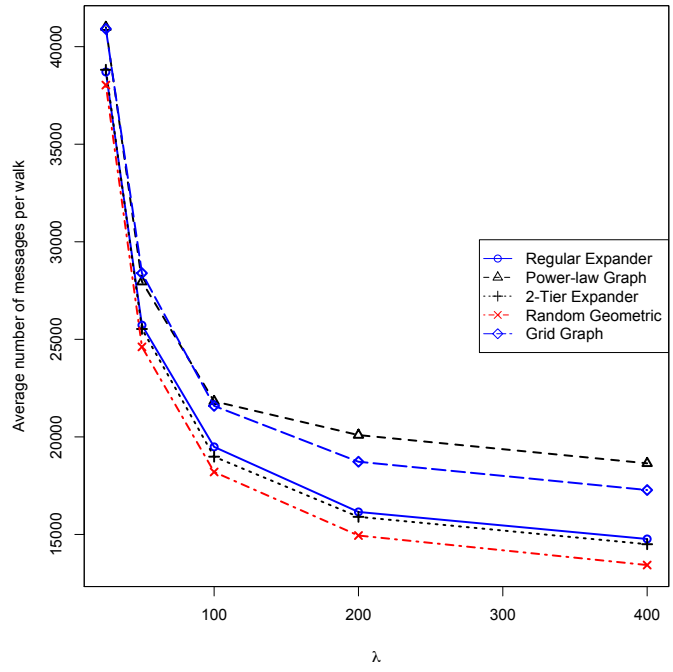


Figure 2.8: Varying length of short walk λ. $n = 10K, \ell = n, \eta = 1$

Summary of observed message complexity: In the naive approach, while each random walk requires $O(\ell)$ messages, the round complexity is increased significantly. At the other extreme, each random walk in [42] was round-efficient but required $\Omega(m)$ messages! Our algorithm of Continuous-Random-Walk achieves the best of both worlds by guaranteeing best-possible message and round complexity for graph topologies. The experiments suggest that for a wide range of parameters, the algorithm is able to answer each Single-Random-Walk request in a continuous manner with very few messages or rounds. These results corroborate our theoretical guarantees and highlight the practicality of our technique.

2.5 Conclusion

We present near-optimal distributed algorithms for random walk sampling in networks. Our algorithms are fully decentralized, lightweight, and easily implementable. Since node sampling is useful in various networking applications, our algorithms can serve as building blocks in a variety of distributed networking applications.

Chapter 3

Random Walks in Dynamic Networks and Applications

In this chapter, we study random walks in *dynamic* networks in which the network topology changes (arbitrarily) from round to round[1]. More precisely, we investigate efficient distributed computation in this *dynamic* network via random walks. The local and lightweight nature of random walks is especially useful for providing uniform and efficient solutions to distributed control of dynamic networks. Given their applicability in dynamic networks, we focus on developing fast distributed algorithms for performing random walks in such networks.

We first present a rigorous framework for design and analysis of distributed random walk algorithms in dynamic networks. We then develop a fast distributed random walk algorithm that runs in $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ rounds (with high probability), where $\mathcal{T}$ is the *dynamic mixing time* and $\mathcal{D}$ is the *dynamic diameter* of the network respectively, and returns a sample close to a suitably defined stationary distribution of the dynamic network.

Then we present a fast distributed algorithm for the fundamental problem of *information dissemination* (also called as *gossip*) in dynamic networks. In

[1]This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan (which appeared in Symposium on Distributed Computing 2012) and contains material from [36].

gossip, or more generally, k-gossip, there are k pieces of information (or tokens) that are initially present in some nodes and the problem is to disseminate the k tokens to all nodes. We present a random-walk based algorithm that runs in $\tilde{O}(\min\{n^{1/3}k^{2/3}(\mathcal{T}\mathcal{D})^{1/3}, k\mathcal{D}\})$ rounds (with high probability).

3.1 Introduction

Random walks are useful in networks in various ways. The local and lightweight nature of random walks make them especially attractive to dynamic networks. They are particularly useful in providing uniform and efficient solutions to distributed control of dynamic networks [23, 93]. We show random walks as a powerful tool to design distributed algorithms for dynamic networks. For this, we focus on two main goals of this chapter: (1) design fast distributed algorithms for performing random walk sampling efficiently in dynamic networks, and (2) applying random walks as a key subroutine to solve non-trivial distributed computation problems in dynamic networks.

3.1.1 Outline of This Chapter

In the next section (Section 3.1.2) we formally define the dynamic graph model, distributed computing model we consider and some important parameters in dynamic network. Section 3.2 builds the random walk framework in dynamic networks, and also defines the associated parameters. Section 3.3 formally states the problems and presents our results. Section 3.4 talks about related work on dynamic networks and random walks and gives a technical overview. In Section 3.5, we briefly discuss subsequent works which use our random work framework in dynamic networks. The distributed random walk algorithm for single random walk and the precise theorem statements are in Section 3.6; the corresponding results for k random walks are in Section 3.7. Information dissemination is discussed in Section 3.8. We conclude with a summary in Section 3.9.

3.1.2 Network Model and Definitions

Dynamic Networks

We study a general model to describe a dynamic network with a *fixed* set of nodes. We consider an oblivious adversary which can make *arbitrary* changes to the graph topology in every round as long as the graph is *connected*. Such a dynamic graph process (or dynamic graph, for short) is also known as an *Evolving Graph* [9]. Suppose $V = \{v_1, v_2, \ldots, v_n\}$ be the set of nodes (vertices) and $\mathcal{G} = G_1, G_2, \ldots$ be an infinite sequence of undirected (connected) graphs on V. We write $G_t = (V, E_t)$ where $E_t \in 2^{V \times V}$ is the dynamic edge set corresponding to round $t \in \mathbb{N}$. The adversary has complete control on the topology of the graph at each round, however it does not know the random choices made by the algorithm. In particular, in the context of random walks, we assume that it does not know the position of the random walk in any round (however, the adversary may know the starting position)[2]. Equivalently, we can assume that the adversary chooses the entire sequence $\langle G_t \rangle$ of the graph process $\mathcal{G}$ in advance before execution of the algorithm. This adversarial model has also been used in [9] in their study of random walks in dynamic networks.

This model captures most interesting scenarios of dynamic networks, but most natural problems are NP-complete such as finding strongly connected components and the equivalence of minimum spanning tree [9, 47]. We say that the dynamic graph process $\mathcal{G}$ has some property when each G_t has that property. We assume that each graph G_t is connected and bounded degree graph i.e., the degree of any node can be at most d, where d is a constant. However, for now, to make it simpler, we will assume that each graph G_t is d-regular. Later we will show that our results can be easily generalized to apply to non-regular bounded degree

[2]Indeed, an adaptive adversary that always knows the current position of the random walk can easily choose graphs in each step, so that the walk never really progresses to all nodes in the network.

graphs as well with only a constant factor slowdown[3] (cf. Section 3.6.3). Also we will assume that each G_t is non-bipartite. The assumption on non-bipartiteness ensures that the mixing time is well defined, however this restriction can be removed using a standard technique: adding self-loops on each vertices (e.g., see [9]). Henceforth, we assume that the dynamic graph is a *d-regular dynamic graph* unless otherwise stated (these two terms will be used interchangeably).

Model of Computation

We consider the CONGEST model of distributed computation, described in Section 1.2. The communication network is an n-node dynamic graph process $\mathcal{G} = G_1, G_2, \ldots G_t, \ldots$. Every node has limited initial knowledge such as its own identity number and the identity numbers of its neighbors in G_1. We further assume that all nodes know n. The nodes are allowed to communicate through the edges of the graph G_t in each round t in synchronous *rounds*. In particular, at the beginning of each round t, each node v is allowed to send a message of size B bits through each edge $e = (v, u) \in E_t$ that is adjacent to v. For the sake of simplifying our analysis, we assume that $B = O(\log^3 n)$, although this is generalizable.[4]

While there are several measures of efficiency of distributed algorithms (cf. Section 1.2), in this chapter we will focus only on *running time*, i.e. the number of *rounds* of distributed communication.

Another key parameter affecting the efficiency of distributed computation in a dynamic graph is its dynamic diameter (also called flooding time, e.g., see [15, 27]). The *dynamic diameter* (denoted by $\mathcal{D}$) of an n-node dynamic graph

[3] In fact, our results can be generalized to apply to any graphs, albeit at the cost of slower running time.

[4] It turns out that the per-round congestion in any edge in our random walk algorithm is $O(\log^3 n)$ bits w.h.p. Hence assuming this bound for B ensures that the random walks can never be delayed due to congestion. This simplifies the correctness proof of our random walk algorithm (cf. Section 3.6.2.)

$\mathcal{G}$ is the worst-case time (number of rounds) required to broadcast a piece of information from any given node to all n-nodes. The worst-case is taken over all the times starting from any instance G_t to $G_{t_{max}}$, where t_{max} is polynomial in n. In particular, we consider $t = 1$ (i.e., starting from the beginning) and $t_{max} = O(n^3)$, since the worst-case mixing time of any graph is $O(n^3)$. The dynamic diameter can be much larger than the diameter (D) of any (individual) graph G_t, however, it is bounded by n if the dynamic graph is connected.

3.2 Random Walks in a Dynamic Graph

In this section, we formalize the notion of random walk in a dynamic graph and define and prove bounds on the dynamic mixing time.

Throughout this chapter, we assume the *simple random walk* in an undirected graph: In each step, the walk goes from the current node to a random neighbor, i.e., from the current node v, the probability to move in the next step to a neighbor u is $\Pr(v, u) = 1/d(v)$ for $(v, u) \in E$ and 0 otherwise ($d(v)$ is the degree of v).

A *simple random walk* on dynamic graph $\mathcal{G}$ is defined as follows: assume that at time t the walker is at node $v \in V$, and let $N(v)$ be the set of neighbors of v in G_t, then the walker goes to one of its neighbors from $N(v)$ uniformly at random.

Suppose we have a random walk $v_0 \to v_1 \to \ldots \to v_t$ on a dynamic graph $\mathcal{G}$, where v_0 is the starting vertex. Then we get a probability distribution P_t on v_t starting from the initial distribution P_0 on v_0. We say that the distribution P_r is stationary (or steady-state) for the graph process $\mathcal{G}$ if $P_{t+1} = P_t$ for all $t \geq r$. It is known that for every (undirected) static graph G, the distribution $\pi(v) = d(v)/2m$ is stationary. In particular, for a regular graph the stationary distribution is the uniform distribution. The *mixing time* of a random walk on a static graph G is the time t taken to reach "close" to the stationary distribution of the graph. Similar to the static case, for a d-regular dynamic graph, it is easy to verify that the stationary distribution is the uniform distribution. Also, for a

d-regular dynamic graph, the notion of *dynamic mixing time* (formally defined in Section 3.2.1) is similar to the static case and is well defined as we show below. We formally show (cf. Theorem 3.1 in Section 3.2.1) that the dynamic mixing time is bounded by $O(\frac{1}{1-\lambda}\log n)$ rounds, where λ is an upper bound of the second largest eigenvalue in absolute value of the transition matrix of any graph in $\mathcal{G}$. Note that $O(\frac{1}{1-\lambda}\log n)$ is also an upper bound on the mixing time of the graph having λ as its second largest eigenvalue [69] and hence the dynamic mixing time is upper bounded by the worst-case mixing time of any graph in $\mathcal{G}$, which will be (henceforth) denoted by $\mathcal{T}$. Since the second eigenvalue of the transition matrix of any regular graph is bounded by $1 - 1/n^2$ (cf. Corollary 3.2), this implies that $\mathcal{T}$ of a d-regular dynamic graph is bounded by $\tilde{O}(n^2)$ (cf. Section 3.2.1). In general, the dynamic mixing time can be significantly smaller than this bound, e.g., when all graphs in $\mathcal{G}$ have λ bounded from above by a constant (i.e., they are expanders — such dynamic graphs occur in applications e.g., [7, 65]), the dynamic mixing time is $O(\log n)$.

3.2.1 Mixing Time of a Dynamic Graph

Definition 3.1 (Distribution vector)**.** *Let $\pi_x(t)$ define the probability distribution vector reached after t steps when the initial distribution starts with probability* 1 *at node x. Let π denote the stationary distribution vector.*

We define the *dynamic mixing time* of a d-regular dynamic graph $\mathcal{G} = G_1, G_2, \ldots$ as the maximum time taken for a simple random walk starting from any node to reach *close to* the uniform distribution on the vertex set. Therefore the definition of dynamic mixing time is similar to the static case. Let $\mathcal{T}$ be the maximum mixing time of any (individual) graph G_t in $\mathcal{G}$. We show that dynamic mixing time is well defined due to Theorem 3.1 and monotonicity property of distribution vector (cf. Lemma 3.2).

Definition 3.2 (Dynamic Mixing Time)**.** *Define $\mathcal{T}^x(\epsilon)$ (ϵ-near mixing time for source x) as $\mathcal{T}^x(\epsilon) = \min t : ||\pi_x(t) - \pi|| < \epsilon$. Note that $\pi_x(t)$ is the probability*

distribution on the graph G_t in the dynamic graph process $\{G_t : t \geq 1\}$ when the initial distribution ($\pi_x(1)$) starts with probability 1 at node x on G_1. Define $\mathcal{T}^x_{mix}$ (mixing time for source x) $= \mathcal{T}^x(1/2e)$ and $\mathcal{T}_{mix} = \max_x \mathcal{T}^x_{mix}$. The dynamic mixing time is upper bounded by $\mathcal{T} = \max\{$mixing time of all the static graph $G_t : t \geq 1\}$. Notice that $\mathcal{T} \geq \mathcal{T}_{mix}$ in general (follows from Lemma 3.1 and Corollary 3.1).

It is known that a simple random walk on regular, connected, non-bipartite static graph have mixing time $O(\frac{\log n}{1-\lambda_2})$, where λ_2 is the second largest eigenvalue in absolute value of the graph. Interestingly, it turns out that a similar result holds for dynamic graphs as well (as stated in Section 3.1.2, throughout we assume a dynamic graph to be d-regular, connected, and non-bipartite). We show that the mixing time of a simple random walk on a dynamic graph $\mathcal{G} = G_1, G_2, \ldots$ is $O(\frac{\log n}{1-\lambda})$, where λ is an upper bound of the second largest eigenvalue in absolute value of the graphs $\{G_t : t \geq 1\}$.

Lemma 3.1. *([9, 69]) Let G be an undirected, connected, non-bipartite, d-regular graph on n vertices and $p = (p_1, \ldots, p_n)$ be any probability distribution on its vertices. Let A_G be the transition matrix of a simple random walk on G. Then,*

$$\left\|pA_G - \frac{\boldsymbol{1}}{n}\right\| \leq \bar{\lambda_2} \cdot \left\|p - \frac{\boldsymbol{1}}{n}\right\|$$

where $\bar{\lambda_2} = \max_{i=2,\ldots,n}|\lambda_i| = \max\{\lambda_2, -\lambda_n\}$ is the second largest eigenvalue in absolute value.

Proof. Let $X_1, X_2, \ldots, X_n$ be an orthonormal set of eigenvectors of A_G with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Since A_G is symmetric stochastic matrix, $\lambda_1 = 1$ and $X_1 = \frac{\mathbf{1}}{\sqrt{n}}$ and all eigenvectors and eigenvalues are real. Clearly,

$$\left\|pA_G - \frac{\mathbf{1}}{n}\right\| = \left\|pA_G - \frac{\mathbf{1}}{n}A_G\right\| = \left\|(p - \frac{\mathbf{1}}{n})A_G\right\|$$

Since p is a probability distribution, we can write it as $p = \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$, where $\beta_1, \beta_2, \ldots, \beta_n \in \mathbb{R}$. Then $\beta_1 = p \cdot X_1^T = \frac{1}{\sqrt{n}} \sum_i p_i = \frac{1}{\sqrt{n}}$, so that

$\beta_1 X_1 = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$. Therefore, $p - \frac{\mathbf{1}}{n} = \sum_{i=2}^{n} \beta_i X_i$. Hence,

$$\|p - \frac{\mathbf{1}}{n}\| = \sqrt{\sum_{i=2}^{n} \beta_i^2}$$

Furthermore,

$$\begin{aligned}\|(p - \frac{\mathbf{1}}{n})A_G\| &= \|\sum_{i=2}^{n} \beta_i X_i A_G\| = \|\sum_{i=2}^{n} \lambda_i \beta_i X_i\| \\ &= \sqrt{\sum_{i=2}^{n} \lambda_i^2 \beta_i^2} \leq \max_{i=2,\ldots,n} |\lambda_i| \cdot \sqrt{\sum_{i=2}^{n} \beta_i^2} \\ &= \bar{\lambda_2} \cdot \|p - \frac{\mathbf{1}}{n}\|\end{aligned}$$

Thus,

$$\|pA_G - \frac{\mathbf{1}}{n}\| \leq \bar{\lambda_2} \cdot \|p - \frac{\mathbf{1}}{n}\|$$

□

An immediate corollary follows from the previous lemma:

Corollary 3.1. *Let $\mathcal{G} = G_1, G_2, \ldots$ be a sequence of undirected, connected, non-bipartite, d-regular graphs on the same vertex set V. If p_0 is the initial probability distribution on V and we perform a simple random walk on $\mathcal{G}$ starting from p_0, then the probability distribution p_t of the walk after t steps satisfies,*

$$\|p_t - \frac{\boldsymbol{1}}{n}\| \leq \lambda^t \|p_0 - \frac{\boldsymbol{1}}{n}\|$$

where λ is an upper bound on the second largest eigenvalue in absolute value of the graphs $\{G_t : t \geq 1\}$.

Theorem 3.1. *For any d-regular, connected, non-bipartite, dynamic graph $\mathcal{G}$, the dynamic mixing time of a simple random walk on $\mathcal{G}$ is bounded by $O(\frac{1}{1-\lambda} \log n)$, where λ is an upper bound of the second largest eigenvalue in absolute value of any graph in $\mathcal{G}$.*

Proof. Let the random walk start from a given vertex with distribution $p_0 = (1, 0, \ldots, 0)$. From Corollary 3.1 and the fact that $\left\|p_0 - \frac{\mathbf{1}}{n}\right\| = O(1)$ we have,

$$\left\|p_t - \frac{\mathbf{1}}{n}\right\| \leq \lambda^t.$$

For $t = \Theta(\frac{1}{1-\lambda} \log n)$, gives $\left\|p_t - \frac{\mathbf{1}}{n}\right\| \leq \frac{1}{n^{O(1)}}$. □

Corollary 3.2. *([69]) For any d-regular, connected, non-bipartite, dynamic graph $\mathcal{G}$, the dynamic mixing time of a simple random walk on $\mathcal{G}$ is bounded by $O(n^2 \log n)$.*

Proof. This follows from the fact that $(1 - \frac{1}{n^2})$ is an upper bound of the second largest eigenvalue $\bar{\lambda_2}$ of the transition matrix of any undirected connected regular graph on n-vertices. Let G be an undirected connected d-regular graph on n vertices and A_G be the transition matrix of G. Suppose $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ are the eigenvalues of A_G, then we show that for $i \geq 2, \lambda_i \leq 1 - \frac{1}{n^2}$. Note that $\lambda_1 = 1$. Therefore, the normalized eigenvector corresponding to $\lambda_1 = 1$ is $X_1 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)$. Consider any normalized real eigenvector $X \perp X_1$ with its eigenvalue λ. Hence, $\sum_{i=1}^{n} x_i^2 = 1$ and $\sum_{i=1}^{n} x_i = 0$, where $X = (x_1, x_2, \ldots, x_n)$. This implies that there exist t and s with $x_t > 0 > x_s$ such that at least one of x_t or x_s has absolute value $\geq 1/\sqrt{n}$. Therefore, $x_t - x_s \geq 1/\sqrt{n}$. Let $s = v_1 \to v_2 \to \ldots \to v_k = t$ be the vertices on a shortest path from s to t in G, such a path exist since G is connected. Consider $X(\mathbb{I} - A_G)X^T$. Then,

$$\begin{aligned}
1 - \lambda = X(\mathbb{I} - A_G)X^T &= \frac{1}{d} \cdot \sum_{\{i,j\} \in E(G)} (x_i - x_j)^2 \\
&\geq \frac{1}{d} \cdot \sum_{i=1}^{k-1} (x_{v_i} - x_{v_{i+1}})^2 \\
&\geq \frac{1}{d(k-1)} \cdot \left(\sum_{i=1}^{k-1} (x_{v_i} - x_{v_{i+1}}) \right)^2 \qquad \text{[by Cauchy-Schwarz inequality]} \\
&= \frac{1}{d(k-1)} \cdot (x_s - x_t)^2 \\
&\geq \frac{1}{dDn}
\end{aligned}$$

where $k-1 \leq D$, the diameter of the graph. Now it is a fact that the diameter of any connected regular graph is bounded by $O(\frac{n}{d})$. Hence, $\lambda \leq 1 - \frac{1}{n^2}$. □

3.2.2 Monotonicity property of the distribution vector

Let $\pi_x(t)$ define the probability distribution vector of a simple random walk reached after t steps when the initial distribution starts with probability 1 at node x. Let π denote the stationary distribution vector. We show in the following lemma that the vector $\pi_x(t)$ gets closer to π as t increases.

Lemma 3.2. $||\pi_x(t+1) - \pi|| \leq ||\pi_x(t) - \pi||$.

Proof. We need to show that the definition of mixing times are consistent, i.e. monotonic in t, the walk length of the random walk. Let A be the transition matrix of a simple random walk on a d-regular dynamic graph $\mathcal{G}$ which in fact changes from round to round. The entries a_{ij} of A denotes the probability of transitioning from node i to node j. The monotonicity follows from the fact that for any transition matrix A of any regular graph and for any probability distribution vector p,

$$||(p - \frac{\mathbf{1}}{n})A|| < ||p - \frac{\mathbf{1}}{n}||.$$

This result follows from the above Lemma 3.1 and the fact that $\bar{\lambda_2} < 1$.

Let π be the stationary distribution of the matrix A. Then $\pi = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$. This implies that if t is ϵ-near mixing time, then $||pA^t - \pi|| \leq \epsilon$, by definition of ϵ-near mixing time. Now consider $||pA^{t+1} - \pi||$. This is equal to $||pA^{t+1} - \pi A||$ since $\pi A = \pi$. However, this reduces to $||(pA^t - \pi)A|| < ||pA^t - \pi|| \leq \epsilon$. It follows that $(t+1)$ is ϵ-near mixing time and $||pA^{t+1} - \pi|| < ||pA^t - \pi||$. □

3.3 Overview of Our Results

We formally state the problems and our main results.

The Single Random Walk problem: Given a d-regular dynamic graph $\mathcal{G} = (V, E_t)$ and a starting node $s \in V$, our goal is to devise a fast distributed *random walk* algorithm such that, at the end, a destination node, sampled from a $\mathcal{T}$-length walk, outputs the source node's ID (equivalently, one can require s to output the destination node's ID), where $\mathcal{T}$ is (an upper bound on) the dynamic mixing time of $\mathcal{G}$ (cf. Section 3.2.1), under the assumption that $\mathcal{G}$ is modified by an oblivious adversary (cf. Section 3.1.2). Note that this distribution will be "close" to the stationary distribution of $\mathcal{G}$ (stationary distribution and $\mathcal{T}$ are both well-defined — cf. Section 3.2.1). Since we are assuming a d-regular dynamic graph, our goal is to sample from (or close to) the uniform distribution (which is the stationary distribution) using as few rounds as possible. Note that we would like to sample fast via random walk — this is also very important for the applications considered in this chapter. On the other hand, if one had to simply get a uniform random sample, it can be accomplished by other means, e.g., it is easy to obtain it in $O(\mathcal{D})$ rounds (by using flooding).

For clarity, observe that the following naive algorithm solves the above problem in $O(\mathcal{T})$ rounds: The walk of length $\mathcal{T}$ is performed by sending a token for $\mathcal{T}$ steps, picking a random neighbor in each step. Then, the destination node v of this walk outputs the ID of s. Our goal is to perform such sampling with significantly less number of rounds, i.e., in time that is sub-linear in $\mathcal{T}$, in the CONGEST model, and using random walks rather than naive flooding techniques. As mentioned earlier this is needed for the application discussed here. Our result is as follows.

Theorem 3.2. *The algorithm* Single-Random-walk *(cf. Algorithm 4) solves the Single Random Walk problem in a dynamic graph and with high probability finishes in* $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ *rounds.*

The above algorithm assumes that nodes have knowledge of $\mathcal{T}$ (or at least some good estimate of it). (In many applications, it is easy to have a good estimate of $\mathcal{T}$ when there is knowledge of the structure of the individual graphs

— e.g., each G_t is an expanders as in [7, 85].) Notice that in the worst case the value of $\mathcal{T}$ is $\tilde{\Theta}(n^2)$, and hence this bound can be used even if nodes have no knowledge. Therefore putting $\mathcal{T} = \tilde{\Theta}(n^2)$ in the above Theorem 3.2, we see that our algorithm samples a node from the uniform distribution through a random walk in $\tilde{O}(n\sqrt{\mathcal{D}})$ rounds w.h.p. Our algorithm can also be generalized to work for non-regular dynamic graphs also (cf. Section 3.6.3).

We also consider the following extension of the Single Random Walk problem, called the κ *Random Walks problem*: We have κ sources $s_1, s_2, ..., s_\kappa$ and we want each of the κ destinations to output an ID of its corresponding source, assuming that each source initiates an independent random walk of length $\mathcal{T}$. (Equivalently, one can ask each source to output the ID of its corresponding destination.) The goal is to output all the ID's in as few rounds as possible. We show that:

Theorem 3.3. *The algorithm* Many-Random-Walks *(cf. Algorithm 5) solves the κ Random Walks problem in a dynamic graph and with high probability finishes in $\tilde{O}\left(\min\{\sqrt{\kappa\mathcal{T}\mathcal{D}}, \kappa + \mathcal{T}\}\right)$ rounds, where $\kappa = O(\frac{n^2 d^2 \mathcal{D}}{\mathcal{T}})$ and assuming that the source nodes are chosen uniformly at random. If the source nodes are chosen arbitrarily, then we show that the κ Random Walks problem (for any κ) can be solved in $\tilde{O}(\kappa\sqrt{\mathcal{T}\mathcal{D}})$ rounds with high probability.*

Information dissemination (or k-gossip) problem In k-gossip, initially k different tokens are assigned to a set V of $n(\geq k)$ nodes. A node may have more than one token. The goal is to disseminate all the k tokens to all the n nodes. We present a fast distributed randomized algorithm for k-gossip in a dynamic network. Our algorithm uses Many-Random-Walks as a key subroutine; this is the first sub-quadratic time fully-distributed *token forwarding* algorithm. In particular, we present two algorithms for the k-gossip algorithm depending on how the tokens are initially distributed among the (source) nodes.

If k tokens are initially situated among nodes (may not be distinct) arbitrarily then we show the following result.

Theorem 3.4. *In dynamic graphs, the k-gossip problem can be solved with high probability in* $\tilde{O}(\min\{kn^{\frac{1}{2}}(\mathcal{TD})^{\frac{1}{4}}, k\mathcal{D}\})$ *rounds.*

We present a more efficient algorithm if the source nodes (each of which has a token to disseminate) are chosen uniformly at random:

Theorem 3.5. *There is a distributed algorithm that solves the* k-gossip *problem in a dynamic graphs with high probability in* $\tilde{O}(\min\{n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{TD})^{\frac{1}{3}}, k\mathcal{D}\})$ *rounds, assuming that each source node is the origin of one token and the source nodes are chosen uniformly at random.*

3.4 Related Work and Technical Overview

Dynamic networks. As a step towards understanding the fundamental computational power in dynamic networks, recent studies (see e.g., [24, 46, 65, 66] and the references therein) have investigated dynamic networks in which the network topology changes arbitrarily from round to round. In the worst-case model that was studied by Kuhn, Lynch, and Oshman [65], the communication links for each round are chosen by an online adversary, and nodes do not know who their neighbors for the current round are before they broadcast their messages. Unlike prior models on dynamic networks, the model of [65] (like our model) does not assume that the network eventually stops changing; therefore it requires that the *algorithms work correctly and terminate even in networks that change continually over time.*

The work of [9] studied the *cover time* of random walks in an dynamic graph in an oblivious adversarial model. The cover time of a graph is the expected time or number of steps taken by a random walk to visit all the nodes in the graph. In a *regular* dynamic graph, they show that the cover time is always polynomial, while this is not true in general if the graph is not regular — the cover time can be exponential. However, they show that a lazy random walk (i.e., walk with self loops) has polynomial cover time on all graphs. We also use a similar strategy

to show that our distributed random walk algorithms can work on non-regular graphs also, albeit at the cost of an increase in run time. While the work of [9] addressed the cover time of random walks on dynamic graphs, this thesis is concerned with distributed algorithms for computing random walk samples fast with the goal towards applying it to fast distributed computation problems in dynamic networks. The work of [28], studies the flooding time of *Markovian* evolving dynamic graphs, a special class of dynamic graphs.

Distributed random walks. As mentioned earlier that our fast distributed random walk algorithms are based on previous such algorithms designed for *static* networks [42] (cf. Chapter 1). A main contribution of the work in this chapter is showing that building on the approach of [42] yields speed up in random walk computations even in dynamic networks. However, there are some challenging technical issues to overcome in this extension given the continuous dynamic nature. One key technical lemma (called the *Random walk visits Lemma*) that was used to show the almost-optimal run time of $\tilde{O}(\sqrt{\ell D})$ does not directly apply to dynamic networks. In the static setting, this lemma gives a bound on the number of times any node is visited in an ℓ-length walk, for any length that is not much larger than the cover time. More precisely, the lemma states that w.h.p. any node x is visited at most $\tilde{O}(d(x)\sqrt{\ell})$ times, in an ℓ-length walk from any starting node ($d(x)$ is the degree of x). Here, we show that a similar bound applies to an ℓ-length random walk on any d-regular dynamic graph (cf. Lemma 3.6). A key ingredient in the above proof is showing that a technical result due to Lyons [75] can be made to work on a dynamic graph.

Other recent work involving multiple random walks in *static* networks, but in different settings include Alon et. al. [3], Elsässer et. al. [16], and Cooper et al. [31].

Information spreading. The main application of our random walks algorithm is an improved algorithm for information spreading or gossip in *dynamic* networks.

It gives the first sub-quadratic, fully distributed, token forwarding algorithm in dynamic networks, partially answering an open question raised in [46]. Note that the 'sub-quadratic' time is conditional upon some graph parameters such as dynamic mixing time and dynamic diameter. Information spreading is a fundamental primitive in networks which has been extensively studied (see e.g., [46] and the references therein). Information spreading can be used to solve other problems such as broadcasting and leader election. Indeed, solving n-gossip problem, where the number of tokens is equal to the number of nodes in the network, and each node starts with exactly one token, allows any function of the initial states of the nodes to be computed, assuming that the nodes know n [65].

We focus on *token-forwarding* algorithms, which do not manipulate tokens in any way other than storing and forwarding them. Token-forwarding algorithms are simple, often easy to implement, and typically incur low overhead. [65] showed that under their adversarial model, k-gossip can be solved by token-forwarding in $O(nk)$ rounds, but that any deterministic online token-forwarding algorithm needs $\Omega(n \log k)$ rounds. In [46], an almost matching lower bound of $\Omega(nk/\log n)$ is shown.

The above lower bound indicates that one cannot obtain efficient (i.e., sub-quadratic) token-forwarding algorithms for gossip in the adversarial model of [65]. This motivates considering other weaker (and perhaps more realistic) models of dynamic networks.

[46] presented a polynomial-time offline *centralized* token-forwarding algorithm that solves the k-gossip problem on an n-node dynamic network and runs in $O(\min\{nk, n\sqrt{k \log n}\})$ rounds with high probability. This is the first known *sub-quadratic* time token-forwarding algorithm but it is not distributed, and furthermore, the centralized algorithm needs to know the complete evolution of the dynamic graph in advance. It was left open in [46] whether one can obtain a fully-distributed and localized algorithm that also does not know anything about

how the network evolves. In this thesis, we resolve this open question in the affirmative. Our algorithm runs in $\tilde{O}(\min\{n^{1/3}k^{2/3}(\mathcal{T}\mathcal{D})^{1/3}, k\mathcal{D}\})$ rounds with high probability, where $\mathcal{T}$ is dynamic mixing time and $\mathcal{D}$ is dynamic diameter (cf. Chapter 3). This is significantly faster than the $O(k\mathcal{D})$-round algorithm of [65] as well as the above centralized algorithm of [46] when $\mathcal{T}$ is not too large. Note that $\mathcal{D}$ is bounded by $O(n)$ and in regular graphs $\mathcal{T}$ is $O(n^2)$ ($O(n^3)$ in general graphs) and so in general, our bounds cannot be better than $O(k\mathcal{D})$.

We note that an alternative approach based on network coding was due to [54, 55], which achieves an $O(nk/\log n)$ rounds using $O(\log n)$-bit messages (which is not significantly better than the $O(nk)$ bound using token-forwarding), and $O(n+k)$ rounds with large message sizes (e.g., $\Theta(n \log n)$ bits). It thus follows that for large token and message sizes there is a factor $\Omega(\min\{n, k\}/\log n)$ gap between token-forwarding and network coding. We note that in our model we allow only one token per edge per round and thus our bounds hold regardless of the token size. For further references to using network coding for gossip and related problems, we refer to the recent works of [8, 20, 43, 54, 55, 81] and the references therein.

3.5 Overview of Subsequent Work

The random walk framework developed in this chapter has subsequently proved useful in developing robust and efficient algorithms in dynamic networks. We discuss two results below.

Storage and Search in Dynamic Peer-to-Peer (P2P) Networks. In this work [5], we study the problem of storing, maintaining, and searching data in dynamic P2P networks, which experience high adversarial node churn (i.e., nodes can join and leave the network continuously over time). We develop distributed storage and search algorithms which guarantee that a large number of nodes in the network can store, retrieve, and maintain a large number of data items,

despite high node churn rate. The key technical tool used in the above algorithm is random walks. It shows how random walks can be used to derive scalable distributed algorithms in dynamic networks with adversarial node churn. In particular, the efficiency and robustness of the algorithms presented in [5], relies (mainly) on a key technical theorem on random walks in dynamic networks with churn — called the "Soup" Theorem. The theorem says that if all nodes generate tokens and distribute them via random walks in a dynamic network then most tokens do mix (despite large adversarial churn) and have the usual desirable properties as in a static network. The proof of this soup theorem relies on extending our random walk results of this chapter (which do not assume node churn) to dynamic networks with node churn.

Fast Byzantine Agreement in Dynamic Networks. The work of [6] studies dynamic networks with churn in the presence of Byzantine nodes. It presents randomized distributed algorithms that guarantee almost-everywhere Byzantine agreement with high probability under a large number of Byzantine nodes and continuous adversarial churn in a polylogarithmic number of rounds. This work also uses the random walk framework and results developed here. In particular, it shows a "Dynamic Sampling" Theorem which characterizes the properties of random walk tokens in dynamic networks with churn and byzantine nodes.

3.6 Algorithm for Single Random Walk

3.6.1 Description of the Algorithm

We develop an algorithm called Single-Random-Walk (cf. Algorithm 4) for d-regular dynamic graph ($\mathcal{G} = (V, E_t)$). The algorithm performs a random walk of length $\mathcal{T}$ (the dynamic mixing time of $\mathcal{G}$ — cf. Section 3.2) in order to sample a destination from (close to) the uniform distribution on the vertex set V. We

assume that nodes know the dynamic diameter $\mathcal{D}$ and the dynamic mixing time $\mathcal{T}$ in our algorithm.

The high-level idea of the algorithm is to perform "many" short random walks in parallel and later "stitch" the short walks to get the desired walk of length $\mathcal{T}$. In particular, we perform the algorithm in two phases, as follows. For simplicity we call the messages used in Phase 1 as "coupons" and in Phase 2 as "tokens". In Phase 1, we perform $d \log n$ (d is degree of the graph) "short" (independent) random walks of length λ (to bound the running time correctly, we show later that we do short walks of length approximately λ, instead of λ) from each node v, where λ is a parameter whose value will be fixed in the analysis. This is done simply by forwarding $d \log n$ "coupons" having the ID of v from v (for each node v) for λ steps via random walks.

In Phase 2, starting at source s, we "stitch" (see Figure 3.1) some of short walks prepared in Phase 1 together to form a longer walk. The algorithm starts from s and randomly picks one coupon distributed from s in Phase 1. We now discuss how to sample one such coupon randomly and go to the destination vertex of that coupon. This can be done easily as follows: In the beginning of Phase 1, each node v assigns a coupon number for each of its $d \log n$ coupons. At the end of Phase 1, the coupons originating at s (containing ID of s plus a coupon number) are distributed throughout the network (after Phase 1). When a coupon needs to be sampled, node s chooses a random coupon number C (from the unused set of coupons) and informs the destination node (which will be the next stitching point) holding the coupon C through flooding.

Let C be the sampled coupon and v be the destination node of C. s then sends a "token" to v (through flooding) and s deletes coupon C (so that C will not be sampled again next time at s, otherwise, randomness will be destroyed). The process then repeats. That is, the node v currently holding the token samples one of the coupons it distributed in Phase 1 and forwards the token to the destination of the sampled coupon, say v'. Nodes v, v' are called "connectors" — they are

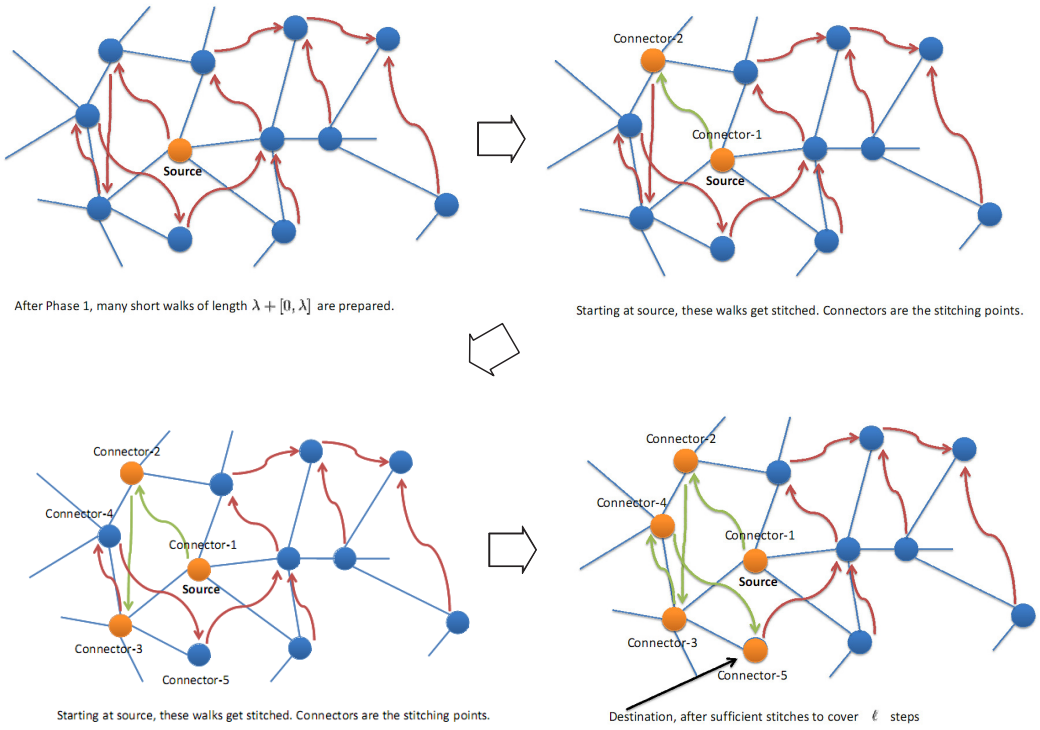


Figure 3.1: Figure illustrating the Algorithm of stitching short walks together. (Figure is taken from [42]).

the endpoints of the short walks that are stitched. A crucial observation is that the walk of length λ used to distribute the corresponding coupons from s to v and from v to v' are independent random walks. Therefore, we can stitch them to get a random walk of length 2λ. We therefore can generate a random walk of length $3\lambda, 4\lambda, \ldots$ by repeating this process. We do this until we have completed more than $\mathcal{T} - \lambda$ steps. Then, we complete the rest of the walk by doing the naive random walk algorithm.

To understand the intuition behind this algorithm, let us analyze its running time. First, we claim that Phase 1 needs $O(\lambda)$ rounds with high probability.

Recall that, in Phase 1, each node prepares $d \log n$ independent random walks of length λ (approximately). We start with $d \log n$ coupons from each node v at the same time, each edge in the current graph should receive $2 \log n$ coupons in the average case. In other words, at most $\log n$ coupons are sent through the same edge. Therefore sending out (just) $d \log n$ coupons from each node for λ steps will take $O(\lambda)$ rounds in expectation in our model. This argument can be modified to show that we need $O(\lambda)$ rounds with high probability (see full proof of the Lemma 3.4). Now by the definition of dynamic diameter, flooding takes $\mathcal{D}$ rounds. We show that sampling a coupon can be done in $O(\mathcal{D})$ rounds (cf. Lemma 3.5) and it follows that Phase 2 needs $\tilde{O}(\mathcal{D} \cdot \mathcal{T}/\lambda)$ rounds. Therefore, the algorithm needs $\tilde{O}(\lambda + \mathcal{D} \cdot \mathcal{T}/\lambda)$ which is $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ when we set $\lambda = \sqrt{\mathcal{T}\mathcal{D}}$.

The reason the above algorithm for Phase 2 is incomplete is that it is possible that $d \log n$ coupons are not enough: We might forward the token to some node v many times in Phase 2 and all coupons distributed by v in the first phase are deleted. In other words, v is chosen as a connector node many times, and all its coupons have been exhausted. If this happens then the stitching process cannot progress. To fix this problem, we will show (in the next section) an important property of the random walk which says that a random walk of length $O(\mathcal{T})$ will visit each node v at most $\tilde{O}(d\sqrt{\mathcal{T}})$ times (cf. Lemma 3.6). But this bound is not enough to get the desired running time, as it does not say anything about the distribution of the connector nodes when we chop the length $\mathcal{T}$ walk into $\mathcal{T}/\lambda$ pieces. There might be some periodicity that results in the same node being visited multiple times but exactly at λ-intervals. We use the following idea to overcome it: Instead of nodes performing walks of length λ, each such walk i does a walk of length $\lambda + r_i$ where r_i is a random number in the range $[0, \lambda - 1]$. Since the random numbers are independent for each walk, each short walks are now of a random length in the range $[\lambda, 2\lambda - 1]$. This modification is needed to claim that each node will be visited as a connector only $\tilde{O}(d\sqrt{\mathcal{T}}/\lambda)$ times (cf. Lemma 3.10). This implies that each node does not have to prepare too many short

Algorithm 4 SINGLE-RANDOM-WALK

Input: Starting node s, desired walk length $\mathcal{T}$ and dynamic diameter $\mathcal{D}$.

Output: Destination node of the walk outputs the ID of s.

Phase 1: (Each node v performs $d \log n$ random walks ($d = \deg(v)$) of length $\lambda + r_i$ where r_i (for each $1 \leq i \leq d \log n$) is chosen independently at random in the range $[0, \lambda - 1]$. At the end of the process, there are $d \log n$ (not necessarily distinct) nodes holding a "coupon" containing the ID of v.)

1: **for** each node v **do**

2: Generate $d \log n$ random integers in the range $[0, \lambda - 1]$, denoted by $r_1, r_2, \ldots, r_{d \log n}$.

3: Construct $d \log n$ messages containing its ID, a counter number and in addition, the i-th message contains the desired walk length of $\lambda + r_i$. We will refer to these messages created by node v as "coupons created by v".

4: **end for**

5: **for** $i = 1$ to 2λ **do**

6: This is the i-th round. Each node v does the following: Consider each coupon C held by v which is received in the $(i-1)$-th round. If the coupon C's desired walk length is at most i, then v keeps this coupon (v is the desired destination). Else, v picks a neighbor u uniformly at random for each coupon C and forwards C to u.

7: **end for**

Phase 2: (Stitch short walks by token forwarding. Stitch $\Theta(\mathcal{T}/\lambda)$ walks, each of length in $[\lambda, 2\lambda - 1]$.)

1: The source node s creates a message called "token" which contains the ID of s.

2: The algorithm will forward the token around and keep track of a set of connectors, denoted by C. Initially, $C = \{s\}$.

{algorithm continued...}

3: **while** Length of walk completed is at most $\mathcal{T} - 2\lambda$ **do**
4: Let v be the node that is currently holding the token.
5: v samples one of the coupons distributed by v uniformly at random (by randomly choosing one counter number from the unused set of coupons). Let v' be the destination node of the sampled coupon, say C.
6: v sends the token to v' through broadcast and deletes the coupon C.
7: $C = C \cup \{v\}$
8: **end while**
9: Walk naively until $\mathcal{T}$ steps are completed (this is at most another 2λ steps)
10: A node holding the token outputs the ID of s

walks. It turns out that this aspect requires quite a bit more work in the dynamic setting and therefore needs new ideas and techniques. The compact pseudo code is given in Algorithm 4.

3.6.2 Analysis

We first show the correctness of the algorithm and then analyze its time complexity.

Correctness

Lemma 3.3. *The algorithm* Single-Random-Walk, *with high probability, outputs a node sample that is close to the uniform probability distribution on the vertex set V.*

Proof. We know (from Theorem 3.1) that any random walk on a regular dynamic graph reaches "close" to the uniform distribution at step $\mathcal{T}$ regardless of any changes of the graph in each round as long as it is d-regular, non-bipartite and connected. Therefore it is sufficient to show that Single-Random-Walk

finishes with a node v which is the destination of a true random walk of length $\mathcal{T}$ on some appropriate dynamic graph from the source node s. We show this below in two steps.

First we show that each short walk (of length approximately λ) created in phase 1 is a true random walk on a dynamic graph sequence $G_1, G_2, \ldots, G_{\tilde{\lambda}}$ ($\tilde{\lambda}$ is some approximate value of λ). This means that in every step t, each walk moves to some random neighbor from the current node on the graph G_t and each walk is independent of others. The proof of the Lemma 3.4 shows that w.h.p there is at most $O(\log^3 n)$ bits congestion in any edge in any round in Phase 1. Since we consider *CONGEST*$(\log^3 n)$ model, at each round $O(\log^3 n)$ bits can be sent through each edge from each direction. Hence effectively there will be no delay in Phase 1 and all walks can extend their length from i to $i+1$ in one round. Clearly each walk is independent of others as every node sends messages independently in parallel. This proves that each short walk (of a random length in the range $[\lambda, 2\lambda - 1]$) is a true random walk on the graph $G_1, G_2, \ldots, G_{\tilde{\lambda}}$.

In Phase 2, we stitch short walks to get a long walk of length $\mathcal{T}$. Therefore, the $\mathcal{T}$-length random walk is not from the dynamic graph sequence $G_1, G_2, \ldots, G_{\mathcal{T}}$; rather it is from the sequence:

$G_1, G_2, \ldots, G_{\tilde{\lambda}}, G_1, G_2, \ldots, G_{\tilde{\lambda}}, \ldots,$ ($\mathcal{T}/\lambda$ times approximately). The stitching part is done on the graph sequence from $G_{\tilde{\lambda}+1}, G_{\tilde{\lambda}+2}, \ldots$ onwards. This does not affect the distribution of probability on the vertex set in each step, since the graph sequence from $G_{\tilde{\lambda}+1}, G_{\tilde{\lambda}+2}, \ldots$ is used only for communication. Also note that since we define $\mathcal{T}$ to be the maximum of any static graph G_t's mixing time, it clearly reaches close to the uniform distribution after $\mathcal{T}$ steps of walk in the graph sequence $G_1, G_2, \ldots, G_{\tilde{\lambda}}, G_1, G_2, \ldots, G_{\tilde{\lambda}}, \ldots,$ ($\mathcal{T}/\lambda$ times approximately).

Finally, when we stitch at a node v, we are sampling a coupon (short walk) uniformly at random among many coupons (and therefore, short walks starting at v) distributed by v. It is easy to see that this stitches short random walks independently and hence gives a true random walk of longer length. Thus it

follows that the algorithm Single-Random-Walk returns a destination node of a $\mathcal{T}$-length random walk (starting from s) on some dynamic graph. □

Time Analysis

We show the running time of algorithm Single-Random-Walk (cf. Theorem 3.2) using the following lemmas. Before going into the detailed proof, we give a roadmap on how it goes. First we show that in the CONGEST model, each node can perform $d \log n$ short random walks of length λ in parallel in $O(\lambda)$ rounds w.h.p. Then it is easy to see that stitching two short walks can be done in $O(\mathcal{D})$ rounds, simply through flooding. Recall that there are approximately $\mathcal{T}/\lambda$ stitches to compute a walk of length $\mathcal{T}$ and it follows that Phase 2 needs $\tilde{O}(\mathcal{D}\mathcal{T}/\lambda)$ rounds. Hence, overall time needed is $\tilde{O}(\lambda+\mathcal{D}\mathcal{T}/\lambda)$ rounds. Therefore, choosing $\lambda = \sqrt{\mathcal{T}\mathcal{D}}$ gives the running time of algorithm Single-Random-Walk as $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ rounds w.h.p. Now the main issue is whether this $d \log n$ short walks from each node would be sufficient to complete the stitching process. We show a technical result (Lemma 3.6) that bounds the number of visits (at most $\tilde{O}(d(x)\sqrt{\ell})$ times) to each node x in a random walk of length ℓ. For this proof, we use the Lyons lemma (see Lemma 3.4 in [75]), which also holds in the regular dynamic graph. Then we show that in a d-regular dynamic graph, no node is visited more than $\tilde{O}(d\sqrt{\mathcal{T}}/\lambda)$ times as a connector node of a $\mathcal{T}$-length random walk. Hence $d \log n$ short walks from each node would be sufficient.

Lemma 3.4. *Phase 1 finishes in $O(\lambda)$ rounds with high probability.*

Proof. In phase 1, each node v performs $d \log n$ walks of length λ. Initially all the nodes start with $d \log n$ coupons (or messages) and each coupon takes a random walk. We prove that after any given number of steps j, the expected number of coupons at node any v is still $d \log n$. At any round, every node has d-neighbors connected with it. So at each step every node can send (as well as receive) d messages. Now the number of messages started at any node v is proportional to its degree and its stationary distribution (which is uniform). Therefore, in

expectation the number of messages at any node remains the same. Thus in expectation the number of messages, say X that go through an edge in any round is at most $2\log n$ (from both end points). Using Chernoff's bound we get ($\Pr[X \geq 4\log^2 n] \leq 2^{-4\log n} = n^{-4}$). It follows that the number of messages can go through any edge in any round is at most $4\log^2 n$ with high probability. Hence there will be at most $O(\log^3 n)$ bits w.h.p. in any edge per round . Since we consider *CONGEST*$(\log^3 n)$ model, so there will be no delay due to congestion. Hence, phase 1 finishes in $O(\lambda)$ rounds with high probability. □

Lemma 3.5. *Sample-Coupon always finishes within $O(\mathcal{D})$ rounds where $\mathcal{D}$ is the dynamic diameter of the network.*

Proof. The proof follows directly from the fact that through flooding one can inform a message to all other nodes in the network. The flooding finishes in diameter time. □

We note that the adversary can force the random walk to visit any particular vertex several times. Then we need many short walks from each vertex which increases the round complexity. We show the following key technical lemma (Lemma 3.6) that bounds the number of visits to each node in a random walk of length ℓ. In a d-regular dynamic graph, we show that no node is visited more than $\tilde{O}(d\sqrt{\mathcal{T}}/\lambda)$ times as a connector node of a $\mathcal{T}$-length random walk. For this we need a technical result on random walks that bounds the number of times a node will be visited in a ℓ-length (where $\ell = O(\mathcal{T})$) random walk. Consider a simple random walk on a connected d-regular dynamic graphs on n vertices. Let $N_t^x(y)$ denote the number of visits to vertex y by time t, given the walk started at vertex x. Now, consider k walks, each of length ℓ, starting from (not necessary distinct) nodes $x_1, x_2, \ldots, x_k$.

Lemma 3.6. (Random Walk Visits Lemma). *For any nodes $x_1, x_2, \ldots, x_k$,*

$$\Pr\big(\exists y \ s.t. \ \sum_{i=1}^{k} N_\ell^{x_i}(y) \geq 32 \ d\sqrt{k\ell+1}\log n + k\big) \leq 1/n\,.$$

To prove the above lemma we need to go through some key auxiliary results. We start with the bound of the first moment of the number of visits at each node by each walk.

Proposition 3.1. *For any node x, node y and $t = O(\mathcal{T})$,*

$$\mathrm{E}[N_t^x(y)] \leq 8\ d\sqrt{t+1} \tag{3.1}$$

To prove the above proposition, let P denote the transition probability matrix of such a random walk and let π denote the stationary distribution of the walk.

The basic bound we use is the estimate from Lyons lemma (see Lemma 3.4 in [75]). We show below that the Lyons lemma also holds for a regular dynamic graph.

Lemma 3.7. *Let Q denote the transition probability matrix of a d-regular dynamic graph, with self-loop probability $\alpha > 0$.*
Let $c = \min\{\pi(x)Q(x,y) : x \neq y \text{ and } Q(x,y) > 0\} > 0$. Note that here $c = \frac{1}{nd}$, as π is uniform distribution. Then for any vertex x and all $k > 0$, a positive integer (denoting time),

$$\Big|\frac{Q^k(x,x)}{\pi(x)} - 1\Big| \leq \min\Big\{\frac{1}{\alpha c\sqrt{k+1}}, \frac{1}{2\alpha^2 c^2(k+1)}\Big\}.$$

Proof. Our approach follows the proof of Lyons lemma (Lemma 3.4 in [75]).

Let $G = (V, E)$ be any d-regular graph and Q be the transition probability matrix of it. Write

$$c_2(x,y) := \pi(x)Q^2(x,y)$$

and note that for $(x,y) \in E$, we have

$$c_2(x,y) \geq \pi(x)\big[Q(x,x)Q(x,y) + Q(x,y)Q(y,y)\big] \geq 2\alpha c.$$

We write $\ell^2(V,\pi)$ for the vector space $\mathbb{R}^V$ equipped with the inner product defined by

$$(f_1, f_2)_\pi := \sum_{x\in V} f_1(x)f_2(x)\pi(x).$$

We regard elements of $\mathbb{R}^V$ as functions from V to $\mathbb{R}$. Therefore we will call eigenvectors of the matrix Q as eigenfunctions. Recall that the transition matrix Q is reversible with respect to the stationary distribution π. The reason for introducing the above inner product is due to the following claim.

Claim 3.1. *Let Q be a reversible transition matrix with respect to π. Then the inner product space $\langle \ell^2(V,\pi), (\cdot,\cdot)_\pi \rangle$ has an orthonormal basis of real-valued eigenfunctions $\{f_i\}_{j=1}^{|V|}$ corresponding to real eigenvalues $\{\lambda_j\}$.*

Proof. Denote by $(\cdot,\cdot)$ the usual inner product on $\mathbb{R}^V$, given by $(f_1, f_2) := \sum_{x\in V} f_1(x)f_2(x)$. For a regular graph, Q is symmetric. The more general proof is given in Lemma 12.2 in [69] where Q need not be symmetric. The spectral theorem for symmetric matrices guarantees that the inner product space $\langle \ell^2(V,\pi), (\cdot,\cdot) \rangle$ has an orthonormal basis $\{\varphi_j\}_{j=1}^{|V|}$ such that φ_j is an eigenfunction with the real eigenvalue λ_j. It is known that $\sqrt{\pi}$ is an eigenfunction of Q corresponding to the eigenvalue 1; we set $\varphi_1 = \sqrt{\pi}$ and $\lambda_1 = 1$. If D_π denote the diagonal matrix with diagonal entries $D_\pi(x,x) = \pi(x)$, then $Q = D_\pi^{\frac{1}{2}} Q D_\pi^{-\frac{1}{2}}$. Let $f_j = D_\pi^{-\frac{1}{2}}\varphi_j$, then f_j is an eigenfunction of Q with eigenvalue λ_j. In fact:

$$Qf_j = QD_\pi^{-\frac{1}{2}}\varphi_j = D_\pi^{-\frac{1}{2}}(D_\pi^{\frac{1}{2}} Q D_\pi^{-\frac{1}{2}})\varphi_j = D_\pi^{-\frac{1}{2}} Q\varphi_j = D_\pi^{-\frac{1}{2}}\lambda_j\varphi_j = \lambda_j f_j$$

Although the eigenfunctions $\{f_j\}$ are not necessarily orthonormal with respect to the usual inner product, they are orthonormal with respect to the inner product $(\cdot,\cdot)_\pi$:

$$\delta_{ij} = (\varphi_i, \varphi_j) = (D_\pi^{\frac{1}{2}} f_i, D_\pi^{\frac{1}{2}} f_j) = (f_i, f_j)_\pi,$$

the first equality follows since $\{\varphi_j\}$ is orthonormal with respect to the usual inner product. □

Let $\ell_0^2(V,\pi)$ be the orthogonal complement of the constants in $\ell^2(V,\pi)$. Note that $\mathbf{1}$ is an eigenfunction of Q and that $\ell_0^2(V,\pi)$ is invariant under Q. Now we show in the following claim that each f has at least one nonnegative value and at least one nonpositive value, such as $f \in \ell_0^2(V,\pi)$.

Claim 3.2. *Let G be an undirected connected d-regular graph on n vertices with transition matrix A_G. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be the eigenvalues and $X_1, X_2, \ldots, X_n$ are the corresponding eigenvectors of A_G. Then for each eigenvector $X_i, i = 2, 3, \ldots, n$, (other than X_1), has at least one negative and at least one positive co-ordinate.*

Proof. It is known that $\lambda_1 = 1$ and the normalized eigenvector corresponding to λ_1 is $X_1 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)$. The set of eigenvectors $\{X_i : i = 1, 2, \ldots, n\}$ form a orthonormal basis of the eigenspace. Then for any normalized eigenvector $X \in \{X_i : i = 2, 3, \ldots, n\}$, we have $X \perp X_1$. Hence, $\sum_{i=1}^{n} x_i^2 = 1$ and $\sum_{i=1}^{n} x_i = 0$, where $X = (x_1, x_2, \ldots, x_n)$. This implies that there exist t and s such that $x_t > 0 > x_s$. □

Let x_0 be a vertex where $|f|$ achieves its maximum. Then it follows from the Claim 3.2

$$\begin{aligned} \|f\|_\infty = |f(x_0)| &\leq \frac{1}{2} \sum_{(x,y)\in E} |f(x) - f(y)| \\ &\leq \frac{1}{2} \sum_{x,y\in V} c_2(x, y)|f(x) - f(y)|/(2\alpha c) \end{aligned} \tag{3.2}$$

where $\|\cdot\|_\infty$ denotes the supremum norm and the factor $1/2$ arises from counting each pair (x, y) in each order. Take $f \in \ell_0^2(V, \pi)$. Notice that $\sum_{x,y\in V} c_2(x, y) = \sum_{x\in V} \pi(x) = 1$. Thus, we have from equation (3.2) by using Cauchy-Schwartz inequality that

$$\begin{aligned} (2\alpha c)^2 \|f\|_\infty^2 &\leq \frac{1}{2} \sum_{x,y\in V} c_2(x, y)[f(x) - f(y)]^2 \\ &= \frac{1}{2} \sum_{x,y\in V} f(x)^2 \pi(x) Q^2(x, y) \\ &\quad - \sum_{x,y\in V} f(x) f(y) \pi(x) Q^2(x, y) \\ &\quad + \frac{1}{2} \sum_{x,y\in V} f(y)^2 \pi(x) Q^2(x, y) \end{aligned}$$

By reversibility, $\pi(x)Q^2(x,y) = \pi(y)Q^2(y,x)$, and the first and last terms above are equal to common value

$$\frac{1}{2}\sum_{x\in V} f(x)^2\pi(x)\sum_{y\in V} Q^2(x,y) = \frac{1}{2}\sum_{x\in V} f(x)^2\pi(x).$$

Therefore the above inequality becomes,

$$\begin{aligned}(2\alpha c)^2\|f\|_\infty^2 &\le \sum_{x\in V} f(x)^2\pi(x) - \sum_{x\in V} f(x)\big[\sum_{y\in V} f(y)Q^2(x,y)\big]\pi(x)\\ &= (f,f)_\pi - (f,Q^2f)_\pi\\ &= ((\mathbb{I}-Q^2)f,f)_\pi.\end{aligned}$$

Alternatively, we may apply (3.2) to the function $\mathrm{sgn}(f)f^2$. Using the trivial inequality

$$|\mathrm{sgn}(s)s^2 - \mathrm{sgn}(t)t^2| \le |s-t|\cdot(|s|+|t|),$$

valid for any real numbers s and t, we obtain that

$$\begin{aligned}(2\alpha c)^2\|f\|_\infty^4 &\le \left(\frac{1}{2}\sum_{x,y\in V} c_2(x,y)|f(x)-f(y)|\cdot(|f(x)|+|f(y)|)\right)^2\\ &\le \left(\frac{1}{2}\sum_{x,y\in V} c_2(x,y)[f(x)-f(y)]^2\right)\cdot\left(\frac{1}{2}\sum_{x,y\in V} c_2(x,y)[|f(x)|+|f(y)|]^2\right)\\ &= ((\mathbb{I}-Q^2)f,f)_\pi\cdot((\mathbb{I}+Q^2)|f|,|f|)_\pi\end{aligned}$$

by the Cauchy-Schwartz inequality and same algebra as above. Therefore, if $(f,f)_\pi \le 1$, we have $2(\alpha c)^2\|f\|_\infty^4 \le ((\mathbb{I}-Q^2)f,f)_\pi$.
Putting both these estimates together, we get

$$2(\alpha c)^2\max\{2\|f\|_\infty^2, \|f\|_\infty^4\} \le ((\mathbb{I}-Q^2)f,f)_\pi \tag{3.3}$$

for $(f,f)_\pi \le 1$. Now we show that the above inequality also holds for a regular dynamic graph.

Claim 3.3. *Let $\mathcal{G} = G_1, G_2, \ldots$ be a d-regular, connected dynamic graph with the same set V of nodes. Let A_{G_i} be the transpose of the transition matrix of G_i.*

Let the column vector $f = (p_1, p_2, \ldots, p_n)^T$ *be any probability distribution on* V. *Then*
$\|(A_{G_{i+1}} A_{G_i} \ldots A_{G_1})f\|_\infty \leq \|(A_{G_i} A_{G_{i-1}} \ldots A_{G_1})f\|_\infty$ *for all* $i \geq 1$.

Proof. It is known that the transition matrix of any regular graph is doubly stochastic and if a matrix Q is doubly stochastic then so is Q^2.
Let $(A_{G_i} A_{G_{i-1}} \ldots A_{G_1})f = (p_1^i, p_2^i, \ldots, p_n^i)^T$ and $\|(A_{G_i} A_{G_{i-1}} \ldots A_{G_1})f\|_\infty = \max\{p_l^i : l = 1, 2, \ldots, n\} = |p_k^i|$ (say). Then

$$(A_{G_{i+1}} A_{G_i} \ldots A_{G_1})f = \big(\sum_{j \in N(1)} a_{1j} p_j^i, \sum_{j \in N(2)} a_{2j} p_j^i, \ldots, \sum_{j \in N(n)} a_{nj} p_j^i \big)^T$$

where $N(v)$ is the set of neighbors of v and a_{ij} is the ij-th entries of the matrix $A_{G_{i+1}}$. We show that the absolute value of any co-ordinates of $(A_{G_{i+1}} A_{G_i} \ldots A_{G_1})f$ is $\leq |p_k^i|$. In fact for any l,

$$|\sum_{j \in N(l)} a_{lj} p_j^i| \leq \sum_{j \in N(l)} |a_{lj}||p_j^i| \leq |p_k^i| \sum_{j \in N(l)} a_{lj} = |p_k^i|,$$

since the matrix is doubly stochastic, the last sum is 1. □

Now apply the inequality (3.3) to $Q^l f$ for $l = 0, 1, \ldots, k$. Summing these inequalities and using Claim 3.3 we obtain,

$$\begin{aligned}
(k+1)2(\alpha c)^2 \max\{2\|Q^k f\|_\infty^2, \|Q^k f\|_\infty^4\} &\leq 2(\alpha c)^2 \max\{2\sum_{l=0}^{k} \|Q^l f\|_\infty^2, \sum_{l=0}^{k} \|Q^k f\|_\infty^4\} \\
&\leq \sum_{l=0}^{k} ((\mathbb{I} - Q^2)Q^l f, Q^l f)_\pi \\
&= \sum_{l=0}^{k} ((\mathbb{I} - Q^2)Q^{2l} f, f)_\pi \\
&= ((\mathbb{I} - Q^{2k+2})f, f)_\pi \leq 1
\end{aligned}$$

for $(f, f)_\pi \leq 1$. This shows that the norm of $Q^k : \ell_0^2(V, \pi) \to \ell^\infty(V)$ is bounded by

$$\beta_k := \min\{[(2\alpha c)^2(k+1)]^{-1/2}, [(\alpha c)^2(2k+2)]^{-1/4}\}.$$

Let $T : \ell^2(V,\pi) \to \ell^2_0(V,\pi)$ be the orthogonal projection $Tf := f - (f,\mathbf{1})_\pi \mathbf{1}$. Given what we have shown, we see that the norm of $Q^k T : \ell^(V,\pi) \to \ell^\infty(V)$ is bounded by β_k. By duality, the same bound holds for $TQ^k : \ell^1(V,\pi) \to \ell^2(V,\pi)$. Therefore by composition of mapping we deduce that the norm of $Q^k T Q^k : \ell^1(V,\pi) \to \ell^\infty(V)$ is at most β_k^2 and the norm of $Q^k T Q^{k+1} : \ell^1(V,\pi) \to \ell^\infty(V)$ is at most $\beta_k \beta_{k+1}$. Applying these inequalities to $f := \mathbf{1}_x/\pi(x)$ gives the required bound. □

For $k = O(\mathcal{T})$ and small α, the above can be simplified to the following bound; see Remark 3 in [75].

$$Q^k(x,y) \le \frac{4\pi(y)}{c\sqrt{k+1}} = \frac{4d}{\sqrt{k+1}}\,. \tag{3.4}$$

Note that given a simple random walk on a graph G, and a corresponding matrix P, one can always switch to the lazy version $Q = (I+P)/2$, and interpret it as a walk on graph G', obtained by adding self-loops to vertices in G so as to double the degree of each vertex. In the following, with abuse of notation we assume our P is such a lazy version of the original one.

Proof of Proposition 3.1. Remember that the dynamic graph is $\mathcal{G} = G_1, G_2, \ldots$. Let $X_0, X_1, \ldots$ describe the random walk, with X_i denoting the position of the walk at time $i \ge 0$ on G_{i+1}, and let $\mathbf{1}_A$ denote the indicator (0-1) random variable, which takes the value 1 when the event A is true. In the following we also use the subscript x to denote the fact that the probability or expectation is with respect to starting the walk at vertex x. First the expectation.

$$\begin{aligned}
\mathrm{E}[N_t^x(y)] = \mathrm{E}_x[\sum_{i=0}^{t} \mathbf{1}_{\{X_i=y\}}] &= \sum_{i=0}^{t} P^i(x,y) \\
&\le 4d \sum_{i=0}^{t} \frac{1}{\sqrt{i+1}}, \quad \text{(using the above inequality (3.4))} \\
&\le 8d\sqrt{t+1}\,.
\end{aligned}$$

□

Using the above proposition, we bound the number of visits of each walk at each node, as follows.

Lemma 3.8. *For $t = O(\mathcal{T})$ and any vertex $y \in \mathcal{G}$, the random walk started at x satisfies:*

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

Proof. First, it follows from the Proposition that

$$\Pr\big(N_t^x(y) \geq 4 \cdot 8\ d\sqrt{t+1}\big) \leq \frac{1}{4}\,. \tag{3.5}$$

For any r, let $L_r^x(y)$ be the time that the random walk (started at x) visits y for the r^{th} time. Observe that, for any r, $N_t^x(y) \geq r$ if and only if $L_r^x(y) \leq t$. Therefore,

$$\Pr(N_t^x(y) \geq r) = \Pr(L_r^x(y) \leq t). \tag{3.6}$$

Let $r^* = 32\ d\sqrt{t+1}$. By (3.5) and (3.6), $\Pr(L_{r^*}^x(y) \leq t) \leq \frac{1}{4}$. We claim that

$$\Pr(L_{r^*\log n}^x(y) \leq t) \leq \left(\frac{1}{4}\right)^{\log n} = \frac{1}{n^2}\,. \tag{3.7}$$

To see this, divide the walk into $\log n$ independent subwalks, each visiting y exactly r^* times. Since the event $L_{r^*\log n}^x(y) \leq t$ implies that all subwalks have length at most t, (3.7) follows. Now, by applying (3.6) again,

$$\Pr(N_t^x(y) \geq r^*\log n) = \Pr(L_{r^*\log n}^x(y) \leq t) \leq \frac{1}{n^2}$$

as desired. □

We now extend the above lemma to bound the number of visits of *all* the walks at each particular node.

Lemma 3.9. *For $t = O(\mathcal{T})$, and for any vertex $y \in \mathcal{G}$, the random walk started at x satisfies:*

$$\Pr\big(\sum_{i=1}^{k} N_t^{x_i}(y) \geq 32\ d\sqrt{kt+1}\log n + k\big) \leq \frac{1}{n^2}\,.$$

Proof. First, observe that, for any r,

$$\Pr\Big(\sum_{i=1}^{k} N_t^{x_i}(y) \geq r - k\Big) \leq \Pr[N_{kt}^{y}(y) \geq r].$$

To see this, we construct a walk W of length kt starting at y in the following way: For each i, denote a walk of length t starting at x_i by W_i. Let $\mathcal{T}_i$ and $\mathcal{T}'_i$ be the first and last time (not later than time t) that W_i visits y. Let W'_i be the subwalk of W_i from time $\mathcal{T}_i$ to $\mathcal{T}'_i$. We construct a walk W by stitching $W'_1, W'_2, ..., W'_k$ together and complete the rest of the walk (to reach the length kt) by a normal random walk. It then follows that the number of visits to y by $W_1, W_2, \ldots, W_k$ (excluding the starting step) is at most the number of visits to y by W. The first quantity is $\sum_{i=1}^{k} N_t^{x_i}(y) - k$. (The term '$-k$' comes from the fact that we do not count the first visit to y by each W_i which is the starting step of each W'_i.) The second quantity is $N_{kt}^{y}(y)$. The observation thus follows.

Therefore,

$$\begin{aligned}
&\Pr\Big(\sum_{i=1}^{k} N_t^{x_i}(y) \geq 32\ d\sqrt{kt+1}\log n + k\Big) \\
&\leq \Pr\big(N_{kt}^{y}(y) \geq 32\ d\sqrt{kt+1}\log n\big) \\
&\leq \frac{1}{n^2}
\end{aligned}$$

where the last inequality follows from Lemma 3.8. □

Now the Random Walk Visits Lemma (cf. Lemma 3.6) follows immediately from Lemma 3.9 by union bounding over all nodes.

The above lemma says that the number of visits to each node can be bounded. However, for each node, we are only interested in the case where it is used as a connector (the stitching points). The lemma below shows that the number of visits as a connector can be bounded as well; i.e., if any node appears t times in the walk, then it is likely to appear roughly t/λ times as connectors.

Lemma 3.10. *For any vertex v, if v appears in the walk at most t times then it appears as a connector node at most $t(\log n)^2/\lambda$ times with probability at least $1 - 1/n^2$.*

Proof. Intuitively, this argument is simple, since the connectors are spread out in steps of length approximately λ. However, there might be some periodicity that results in the same node being visited multiple times but exactly at λ-intervals. To overcome this we crucially use the fact that the algorithm uses short walks of length $\lambda + r$ (instead of fixed length λ) where r is chosen uniformly at random from $[0, \lambda - 1]$. Then the proof can be shown via constructing another process equivalent to partitioning the $\mathcal{T}$ steps into intervals of λ and then sampling points from each interval.

The above statement can be analyzed by constructing a different process that stochastically dominates the process of a node occurring as a connector at various steps in the $\mathcal{T}$-length walk and then using a Chernoff bound argument. For complete proof we refer the Lemma 3.13 in [42], however, the proof follows easily from the following two claims.

Claim 3.4. *(Claim 3.14 in [42]) Consider any sequence A of numbers $a_1, ..., a_{T'}$ of length T'. For any integer λ', let B be a sequence $a_{\lambda'+r_1}, a_{2\lambda'+r_1+r_2}, ..., a_{i\lambda'+r_1+...+r_i}, \cdots$ where r_i, for any i, is a random integer picked uniformly from $[0, \lambda'-1]$. Consider another subsequence of numbers C of A where an element in C is picked from from "every λ' numbers" in A; i.e., C consists of $\lfloor T'/\lambda' \rfloor$ numbers $c_1, c_2, ...$ where, for any i, c_i is chosen uniformly at random from $a_{(i-1)\lambda'+1}, a_{(i-1)\lambda'+2}, ..., a_{i\lambda'}$. Then, $Pr[C$ contains $a_{i_1}, a_{i_2}, ..., a_{i_k}\}] = Pr[B = \{a_{i_1}, a_{i_2}, ..., a_{i_k}\}]$ for any set $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$.*

Claim 3.5. *(Claim 3.15 in [42]) Consider any sequence A of numbers $a_1, ..., a_{T'}$ of length T'. Consider subsequence of numbers C of A where an element in C is picked from from "every λ' numbers" in A; i.e., C consists of $\lfloor T'/\lambda' \rfloor$ numbers $c_1, c_2, ...$ where, for any i, c_i is chosen uniformly at random from the set of*

numbers $a_{(i-1)\lambda'+1}, a_{(i-1)\lambda'+2}, ..., a_{i\lambda'}$.. *For any number* x*, let* n_x *be the number of appearances of* x *in* A*; i.e.,* $n_x = |\{i \mid a_i = x\}|$*. Then, for any* $R \geq 6n_x/\lambda'$*,* x *appears in* C *more than* R *times with probability at most* 2^{-R}*.*

Now we use the claim to prove the lemma. Choose $T' = \mathcal{T}$ and $\lambda' = \lambda$ and consider any node v that appears at most t times. The number of times it appears as a connector node is the number of times it appears in the subsequence B described in the claim. By applying the claim with $R = t(\log n)^2$, we have that v appears in B more than $t(\log n)^2$ times with probability at most $1/n^2$. □

Now we are ready to prove the main result (Theorem 3.2) of this section.

Theorem 3.6. *The algorithm* Single-Random-walk *(cf. Algorithm 4) solves the Single Random Walk problem and with high probability finishes in* $\tilde{O}(\sqrt{\mathcal{TD}})$ *rounds.*

Proof. First, we claim, using Lemma 3.6 and 3.10, that each node is used as a connector node at most $\frac{32\ d\sqrt{\mathcal{T}}(\log n)^3}{\lambda}$ times with probability at least $1 - 2/n$. To see this, observe that the claim holds if each node x is visited at most $t(x) = 32\ d\sqrt{\mathcal{T}+1}\log n$ times and consequently appears as a connector node at most $t(x)(\log n)^2/\lambda$ times. By Lemma 3.6, the first condition holds with probability at least $1-1/n$. By Lemma 3.10 and the union bound over all nodes, the second condition holds with probability at least $1 - 1/n$, provided that the first condition holds. Therefore, both conditions hold together with probability at least $1 - 2/n$ as claimed.

Now, we choose $\lambda = 32\sqrt{\mathcal{TD}}(\log n)^2$. By Lemma 3.4, Phase 1 finishes in $O(\lambda) = \tilde{O}(\sqrt{\mathcal{TD}})$ rounds with high probability. For Phase 2, Sample-Coupon is invoked $O(\frac{\mathcal{T}}{\lambda})$ times (only when we stitch the walks) and therefore, by Lemma 3.5, contributes $O(\frac{\mathcal{TD}}{\lambda}) = \tilde{O}(\sqrt{\mathcal{TD}})$ rounds.

Therefore, with probability at least $1 - 2/n$, the rounds are $\tilde{O}(\sqrt{\mathcal{TD}})$ as claimed. □

3.6.3 Generalization to Non-regular Dynamic Graphs

By using a *lazy* random walk strategy, we can generalize our results to work for a non-regular dynamic graph also. The lazy random walk strategy "converts" a random walk on an non-regular graph to a slower random walk on a regular graph.

Definition 3.3. *At each step of the walk pick a vertex v from V uniformly at random and if there is an edge from the current vertex to the vertex v then we move to v, otherwise we stay at the current vertex.*

This strategy of lazy random walk in fact makes the graphs n-regular: every edge adjacent to the current vertex is picked with the probability $1/n$ and with the remaining probability we stay at the current vertex. Using this strategy, we can obtain the same results on any non-regular graphs as well, but with a factor of n slower. However, we can do better, if nodes know an upper bound d_{max} on the maximum degree of the dynamic network. Modify the lazy walk such that at each step, the walk stays at the current vertex u with probability $1 - (d(u)/(d_{max} + 1))$ and with the remaining probability goes to a neighbor chosen uniformly at random. This only results in a slow down by a factor of d_{max} compared to the regular case. Therefore, for a bounded degree dynamic graph where each node's degree is bounded by a constant d, the running time of our algorithm can be affected only by a constant factor.

3.7 Algorithm for κ Random Walks

The previous section was devoted to performing a single random walk of length $\mathcal{T}$ (mixing time) efficiently to obtain a sample from the stationary distribution. In many applications, one typically requires a large number of random walk samples. A larger amount of samples allows for a better estimation of the problem at hand. In this section, we focus on obtaining several random walk samples. Specifically,

we consider the scenario when we want to compute κ independent walks each of length $\mathcal{T}$ from different sources $s_1, s_2, \ldots, s_\kappa$. We show that SINGLE-RANDOM-WALK (cf. Algorithm 4) can be extended to solve this problem.

Algorithm 5 MANY-RANDOM-WALKS

Input: Source nodes $s_1, s_2, \ldots, s_\kappa$ (given from uniform distribution) and desired walks length $\mathcal{T}$ and dynamic diameter $\mathcal{D}$.

Output: Each destination node of the walk outputs the ID of its corresponding source.

Case 1. When $\lambda \geq \mathcal{T}$. [we assumed $\lambda = (32\sqrt{\kappa\mathcal{T}\mathcal{D}+1}\log n + \kappa)(\log n)^2$]

1: Run the naive random walk algorithm, i.e., the sources find walks of length $\mathcal{T}$ simultaneously by sending tokens.

Case 2. When $\lambda < \mathcal{T}$.

Phase 1: (Each node v performs $d\log n$ random walks of length $\lambda + r_i$ where r_i (for each $1 \leq i \leq d\log n$) is chosen independently at random in the range $[0, \lambda - 1]$. At the end of the process, there are $d\log n$ (not necessarily distinct) nodes holding a "coupon" containing the ID of v.)

1: **for** each node v **do**

2: Perform $d\log n$ walks of length $\lambda + r_i$, as in Phase 1 of algorithm SINGLE-RANDOM-WALK.

3: **end for**

Phase 2: (Stitch $\Theta(\mathcal{T}/\lambda)$ short walks for each source node s_j)

1: **for** $j = 1$ to κ **do**

2: Consider source s_j. Use algorithm SINGLE-RANDOM-WALK to perform a walk of length $\mathcal{T}$ from s_j.

3: When algorithm SINGLE-RANDOM-WALK terminates, the sampled destination outputs ID of the source s_j.

4: **end for**

A trivial extension: One immediate approach is to run the Single-Random-Walk algorithm for every source node sequentially. In this case, one can perform as many random walks as needed (i.e., κ can be anything), even from a single source node. The running time of the algorithm will be κ times the running time of the Single-Random-Walk algorithm. Recall that the running time of the Single-Random-Walk algorithm is $\tilde{O}(\sqrt{\mathcal{T}\mathcal{D}})$ rounds. Therefore, in this case the Many-Random-Walks algorithm finishes in $\tilde{O}(\kappa\sqrt{\mathcal{T}\mathcal{D}})$ rounds with high probability.

A faster extension: We can extend the Single-Random-Walk algorithm in a different way to present a faster algorithm if the source nodes are chosen randomly with probability proportional to the node degrees. In particular, the algorithm Many-Random-Walks (pseudocode is given in Algorithm 5) to compute κ walks is essentially repeating the Single-Random-Walk algorithm on each source with one common/shared phase, and yet through overlapping computation, completes faster than the above bound. The crucial observation is that we have to do Phase 1 only once and still ensure all walks are independent. The high level analysis is following.

Many-Random-Walks Let $\lambda = (32\sqrt{\kappa\mathcal{T}\mathcal{D}+1}\log n + \kappa)(\log n)^2$. If $\lambda \geq \mathcal{T}$ then run the naive random walk algorithm. Otherwise, do the following. Do Phase 1 once as before. Modify Phase 2 of Single-Random-Walk to create multiple walks, one at a time; i.e., in the second phase, we stitch the short walks together to get a walk of length $\mathcal{T}$ starting at s_1 then do the same thing for s_2, s_3, and so on. We show that Many-Random-Walks algorithm finishes in $\tilde{O}\left(\min\{\sqrt{\kappa\mathcal{T}\mathcal{D}}, \kappa+\mathcal{T}\}\right)$ rounds with high probability. Moreover, the algorithm is able to perform a constant fraction of $\frac{\lambda n d \log n}{\mathcal{T}}$ walks of length $\mathcal{T}$ if the source nodes are chosen randomly with probability proportional to the node degrees. (Note that it is possible to get $\frac{\lambda n d \log n}{\mathcal{T}}$ random walks, if we can use/stitch all the short walks created in phase 1.) This result is stated in the Theorem 3.3 (Section

3.3), and the formal proof is given below.

Theorem 3.7. *Algorithm* Many-Random-Walks *(cf. Algorithm 5) finishes in* $\tilde{O}\left(\min\{\sqrt{\kappa\mathcal{T}\mathcal{D}}, \kappa+\mathcal{T}\}\right)$ *rounds with high probability, where* $\kappa = O(\frac{n^2d^2\mathcal{D}}{\mathcal{T}})$ *random walks, assuming the source nodes are chosen randomly with probability proportional to the node degrees (for a d-regular dynamic graph, this means that the source nodes are chosen uniformly at random).*

Proof. We first show the correctness of the algorithm. We show that the Many-Random-Walks algorithm samples a node from (close to) the uniform distribution of the vertex set for every source node. In Many-Random-Walks, we create 'short' random walks (i.e., Phase 1) only once. Then for each source node, we stitch those short walks together to get a walk of length $\mathcal{T}$. This is same as repeating the Single-Random-Walk algorithm for each source node. Hence, it follows from the correctness proof of the Single-Random-Walk algorithm that the Many-Random-Walks algorithm samples node from (close to) the uniform distribution for every source node.

Recall that we assume $\lambda = (32\sqrt{\kappa\mathcal{T}\mathcal{D}+1}\log n+\kappa)(\log n)^2$. First, consider the case where $\lambda \geq \mathcal{T}$. In this case, $\tilde{O}(\min\{\sqrt{\kappa\mathcal{T}\mathcal{D}}+\kappa, \sqrt{\kappa\mathcal{T}}+\kappa+\mathcal{T}\}) = \tilde{O}(\sqrt{\kappa\mathcal{T}}+\kappa+\mathcal{T})$. By Lemma 3.6, each node x will be visited at most $\tilde{O}(d(\sqrt{\kappa\mathcal{T}}+\kappa))$ times. Therefore, using the same argument as in proof of Lemma 3.4, the congestion is $\tilde{O}(\sqrt{\kappa\mathcal{T}}+\kappa)$ with high probability. Since the dilation is $\mathcal{T}$, Many-Random-Walks takes $\tilde{O}(\sqrt{\kappa\mathcal{T}}+\kappa+\mathcal{T})$ rounds as claimed. Since $2\sqrt{\kappa\mathcal{T}} \leq \kappa+\mathcal{T}$, this bound reduces to $\tilde{O}(\kappa+\mathcal{T})$.

Now, consider the other case where $\lambda < \mathcal{T}$. In this case, $\tilde{O}(\min\{\sqrt{\kappa\mathcal{T}\mathcal{D}}+\kappa, \sqrt{\kappa\mathcal{T}}+\kappa+\mathcal{T}\}) = \tilde{O}(\sqrt{\kappa\mathcal{T}\mathcal{D}}+\kappa)$. Phase 1 takes $O(\lambda) = \tilde{O}(\sqrt{\kappa\mathcal{T}\mathcal{D}}+\kappa)$. The stitching in Phase 2 takes $\tilde{O}(\kappa\mathcal{T}\mathcal{D}/\lambda) = \tilde{O}(\sqrt{\kappa\mathcal{T}\mathcal{D}})$. Since $\kappa\mathcal{T}\mathcal{D}/\lambda \geq \kappa\mathcal{D} \geq \kappa$, the total number of rounds required is $\tilde{O}(\sqrt{\kappa\mathcal{T}\mathcal{D}})$ as claimed.

We note that each node creates $d\log n$ short walks of length λ in Phase 1.

So there are a total of $nd \log n$ short walks of length λ. Moreover, a short walk can not be reused to get a long walk. We use a technical result shown in [37] that a constant fraction of all the short walks can be utilized (without reusing the same short walk) to create long walks successfully (i.e., the stitching process continues without exhausting short walks of any node) of length $\mathcal{T}$ if the source nodes are chosen randomly proportional to the node degrees. Further, to perform κ random walks of length $\mathcal{T}$, the algorithm must successfully stitch $\kappa\mathcal{T}/\lambda$ short walks. Therefore, if we choose source nodes randomly proportional to the node degrees then $\kappa\mathcal{T}/\lambda$ can be up to a constant fraction of all the short walks, i.e., $\kappa\mathcal{T}/\lambda = \Theta(nd \log n)$. Putting the value of $\lambda = (32\sqrt{\kappa\mathcal{T}\mathcal{D} + 1} \log n + \kappa)(\log n)^2$, we get $\kappa = \tilde{O}(\frac{n^2 d^2 \mathcal{D}}{\mathcal{T}})$. □

3.8 Application: Information Dissemination (or k-Gossip)

While the previous sections focused on performing the fundamental primitive of random walks efficiently in a dynamic network, in this section we show that these techniques directly help in specific applications in dynamic networks as well. We present a fully distributed algorithm for the *k-gossip* problem (defined in Section 3.3) in d-regular dynamic graph (cf. Algorithm 6). Our distributed algorithm is based on the centralized algorithm of [46] which consists of two phases. The first phase consists of sending some f copies (the value of the parameter f will be fixed in the analysis) of each of the k tokens to a set of *random* nodes. We use algorithm Many-Random-Walks (cf. Section 3.7) to efficiently do this. In the second phase we simply broadcast each token t from the random places to reach all the nodes. We show that if every node having a token t broadcasts it for $O(n \log n/f)$ rounds, then with high probability all the nodes will receive the token t.

Algorithm 6 K-INFORMATION-DISSEMINATION

Input: A dynamic graph $\mathcal{G} : G_1, G_2, \ldots$ and k token in some nodes.

Output: To disseminate k tokens to all the nodes.

Phase 1: (Send f copies of each token to random places. f is chosen appropriately in the analysis for two different cases)

1: Every node holding token t, sends f copies of each token to random nodes using the algorithm MANY-RANDOM-WALKS.

Phase 2: (Broadcast each token for $O(n \log n/f)$ rounds)

1: **for** each token t **do**

2: For the next $2n \log n/f$ rounds, let all the nodes that have token t broadcast the token.

3: **end for**

3.8.1 Analysis

First we prove a lemma which will guarantee that our algorithm disseminates the tokens to all the nodes in the network correctly.

Lemma 3.11. *Let $S \subseteq V$ be the set of random nodes chosen from close to the uniform distribution over V. Let all nodes in S contains a token t. If every node having a token t broadcasts it for $O(n \log n/|S|)$ rounds, then all the n nodes receive the token t with high probability.*

Proof. Let $|S| = f$. Fix a node v. Since the token t is broadcast for $2n \log n/f$ rounds, there is a set S_v^t of at least $2n \log n/f$ nodes from which v is reachable within $2n \log n/f$ rounds. This follows from the fact that at any round at least one uninformed node will be informed as the graph is (always) connected. It is now clear that if S intersects S_v^t, v will receive token t. Suppose the elements of the set S were sampled from the vertex set with probability $1/n \pm 1/n^2$, i.e., close to the uniform distribution. So the probability that a single node $w \in S$ does

not intersect S_v^t is at most $(1 - |S_v^t|(\frac{1}{n} \pm \frac{1}{n^2})) = (1 - \frac{2n \log n}{f} \times \frac{n \pm 1}{n^2})$. Therefore the probability that any of the f sampled nodes in S do not intersect S_v^t is at most $(1 - \frac{2(n \pm 1) \log n}{nf})^f \leq \frac{1}{n^{2 \pm 2/n}}$. Now using union bound, it follows that every node in the network receives the token t with high probability. □

Next we analyze the running time of our k-gossip algorithm in two cases.

Case I First we consider the case where k tokens are initially situated among nodes (may not be distinct) arbitrarily. For this, we use the trivial version of Many-Random-Walks algorithm to send the tokens to random places in phase 1. We show that our proposed k-gossip algorithm finishes in $\tilde{O}(kn^{\frac{1}{2}}(\mathcal{T}\mathcal{D})^{\frac{1}{4}})$ rounds w.h.p. To make sure that the algorithm terminates in $O(k\mathcal{D})$ rounds, each node compares the running time of our algorithm and the naive algorithm (which is simply broadcasting each of the k tokens sequentially; clearly this will take $O(k\mathcal{D})$ rounds in total) and run the faster one. (We assume that nodes know the dynamic diameter $\mathcal{D}$ and the dynamic mixing time $\mathcal{T}$.) Moreover, nodes can know the number of tokens k in $O(\mathcal{D})$ rounds. Therefore, each node can easily compare the running time. Thus the claimed bound in Theorem 3.4 holds. The formal proof is given below.

Theorem 3.8. *In dynamic networks, the k-gossip problem can be solved with high probability in $\tilde{O}(\min\{kn^{\frac{1}{2}}(\mathcal{T}\mathcal{D})^{\frac{1}{4}}, k\mathcal{D}\})$ rounds.*

Proof. We run both the naive and our proposed algorithm (cf. Algorithm 6) in parallel. Since the naive algorithm finishes in $O(k\mathcal{D})$ rounds, therefore we concentrate here only on the round complexity of our proposed algorithm.
In Phase 1, we send f copies of each k token to random nodes which means we are sampling kf random nodes from uniform distribution. We assumed that any node may want to disseminate information (even multiple pieces) to all other nodes. Therefore, we use the trivial Many-Random-Walks algorithm which is just repeating the Single-Random-Walk algorithm for each source node. The running time of this trivial Many-Random-Walks algorithm is $\tilde{O}(\kappa\sqrt{\mathcal{T}\mathcal{D}})$ rounds

to perform κ random walks. Hence, Phase 1 of K-Information-Dissemination takes $\tilde{O}(kf\sqrt{\mathcal{TD}})$ rounds to sample kf random nodes.

Now consider a particular token t. Let S be the set of nodes which has the token t after phase 1. Suppose the Many-Random-Walks algorithm samples nodes with probability $1/n \pm 1/n^2$ which means that each node in S is sampled with probability[5] $1/n \pm 1/n^2$. Now we can apply the above Lemma 3.11 and conclude that in phase 2, every node in the network receives the token t with high probability. Therefore, Phase 2 uses $kn \log n/f$ rounds and sends all the k tokens to all the nodes with high probability. Therefore the algorithm finishes in $\tilde{O}(kf\sqrt{\mathcal{TD}} + kn/f)$ rounds. Choosing $f = n^{\frac{1}{2}}/(\mathcal{TD})^{\frac{1}{4}}$ gives the bound as $\tilde{O}(kn^{\frac{1}{2}}(\mathcal{TD})^{\frac{1}{4}})$. Hence, the k-gossip problem can be solved with high probability in $\tilde{O}(\min\{kn^{\frac{1}{2}}(\mathcal{TD})^{\frac{1}{4}}, k\mathcal{D}\})$ rounds. □

Case II Now we consider the case where source nodes (each of which has one token to disseminate) are chosen from a specific distribution. In particular, we assume nodes are chosen randomly proportional to the node degrees, i.e., from the uniform distribution in our model. In this case, we present a more sophisticated algorithm for k-gossip problem. We use the faster Many-Random-Walks algorithm which runs in $\tilde{O}\left(\min\{\sqrt{\kappa\mathcal{TD}}, \kappa + \mathcal{T}\}\right)$ rounds to perform κ random walks (cf. Theorem 3.7).

We show that our proposed k-gossip algorithm finishes in $\tilde{O}(n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{TD})^{\frac{1}{3}})$ rounds w.h.p. Again to make sure that the algorithm terminates in $O(k\mathcal{D})$ rounds, we run the above algorithm in parallel with the naive algorithm and stop when one of the two algorithms stop. Thus the claimed bound in Theorem 3.5 holds. The formal proof is given below.

Theorem 3.9. *In dynamic networks, the k-gossip problem can be solved with high probability in $\tilde{O}(\min\{n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{TD})^{\frac{1}{3}}, k\mathcal{D}\})$ rounds.*

[5]The algorithm Many-Random-Walks samples nodes from close to the uniform distribution and this can be made arbitrarily close to uniform by increasing the length of walk by a suitable constant factor.

Proof. We run both the naive and our proposed algorithm (cf. Algorithm 6) in parallel. Since the naive algorithm finishes in $O(k\mathcal{D})$ rounds, therefore we concentrate here only on the round complexity of our proposed algorithm. Hence we assume that our algorithm has better running time in the following analysis, i.e., we assume $n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{T}\mathcal{D})^{\frac{1}{3}} < k\mathcal{D}$.
In Phase 1, we send f copies of each of the k tokens to random nodes which means we are sampling kf random nodes from uniform distribution. Since the source nodes are chosen uniformly at random, we can use the faster Many-Random-Walks algorithm to efficiently do it. The Many-Random-Walks algorithm can sample a constant fraction of $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$ nodes (cf. Theorem 3.7). Therefore, repeating the Many-Random-Walks algorithm by at most a constant number of times (before starting the Phase 2 of K-Information-Dissemination algorithm), we can sample $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$ nodes. Hence Phase 1 takes $\tilde{O}(\sqrt{kf\mathcal{T}\mathcal{D}})$ rounds and can send at most $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$ tokens in random places.

Now fix a token t. Let S be the set of nodes which has the token t after Phase 1. Then using the Lemma 3.11 as above, we can say that in Phase 2, every node in the network receives the token t with high probability. Therefore, Phase 2 uses $kn\log n/f$ rounds and sends all the k tokens to all the nodes with high probability. Hence the algorithm finishes in $\tilde{O}(\sqrt{kf\mathcal{T}\mathcal{D}} + kn/f)$ rounds. Choosing $f = n^{\frac{2}{3}}k^{\frac{1}{3}}/(\mathcal{T}\mathcal{D})^{\frac{1}{3}}$ gives the bound as $\tilde{O}(n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{T}\mathcal{D})^{\frac{1}{3}})$. Hence, the k-gossip problem can be solved with high probability in $\tilde{O}(\min\{n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{T}\mathcal{D})^{\frac{1}{3}}, k\mathcal{D}\})$ rounds.

The only thing left is to show that $kf < O(\frac{n^2d^2\mathcal{D}}{\mathcal{T}})$. This is because in Phase 1, using Many-Random-Walks algorithm we can sample at most $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$ nodes. Therefore, kf which is $n^{\frac{2}{3}}k^{\frac{4}{3}}/(\mathcal{T}\mathcal{D})^{\frac{1}{3}}$ must be less than $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$. We show that this is true even for $k = n$, the largest value of k. In fact, it is easy to see that $n^{\frac{2}{3}}k^{\frac{4}{3}}/(\mathcal{T}\mathcal{D})^{\frac{1}{3}}$ is less than $\frac{n^2d^2\mathcal{D}}{\mathcal{T}}$ if $\mathcal{T} < d^3\mathcal{D}^2$, assuming $k = n$. As assumed in the beginning of the proof, $n^{\frac{1}{3}}k^{\frac{2}{3}}(\mathcal{T}\mathcal{D})^{\frac{1}{3}} < k\mathcal{D}$, which gives $\mathcal{T} < k\mathcal{D}^2/n$ which is always less than $d^3\mathcal{D}^2$ for any k. □

Remark 3.1. *Our proposed algorithm has better running time than the naive algorithm when* $\mathcal{T} < \mathcal{D}^3/n^2$ *in* Case I. *In* Case II *it is better when* $\mathcal{T} < k\mathcal{D}^2/n$ *(where,* $\mathcal{T}$ *is dynamic mixing time and* $\mathcal{D}$ *is dynamic diameter of the network). Further, we note that a typical version of the* k*-gossip problem where* $k = n$ *and every node contains a token to disseminate to all other nodes in the network can be solved by appealing to* Case II. *In particular, the running time for this version will be within a polylogarithmic factor of the* Case II *running time.*

3.9 Conclusion

We presented fast and fully decentralized algorithms for performing random walks in distributed dynamic networks. Our algorithms satisfy strong round complexity guarantees and the work presents robust techniques for this fundamental graph primitive in dynamic graphs. We further extend the work to show how it can be used for efficient sampling and other applications such as token dissemination. Our bounds for the token dissemination problem improve on previously best known algorithms under a suitably general dynamic graph model. The framework and results have also been used subsequently in other dynamic network applications as well [5, 6].

The work of this chapter opens several interesting research directions. In the recent years, several fundamental graph operatives are being explored in various distributed dynamic models, and it would be interesting to explore further along these lines and obtain new approaches for identifying sparse cuts or graph partitioning, and similar spectral quantities. For instance, it remains open whether the random walk techniques and subsequent bounds presented in this chapter are optimal. Specifically, are the bounds for the information dissemination problem tight? If yes, can one improve upon the bounds for somewhat more restrictive dynamic graph models? These algorithmic ideas may be useful building blocks in designing fully dynamic self-aware distributed graph

systems (where random walk techniques are helpful [93]).

Chapter 4

Distributed PageRank Computation

This chapter focuses on distributed PageRank computation[1]. PageRank is the steady state distribution (or the top eigenvector of the Laplacian) corresponding to a slightly modified random walk process from the standard random walk. In distributed computing, PageRank vector, or more generally random walk based quantities have been used for several different applications ranging from determining important nodes, load balancing, search, and identifying connectivity structures.

We derive fast random walk-based distributed algorithms for computing PageRank in general graphs and prove strong bounds on the round complexity. We first present a distributed algorithm that takes $O(\log n/\epsilon)$ rounds with high probability on any graph i.e., directed or undirected, where n is the network size and ϵ is the reset probability used in the PageRank computation (typically ϵ is a fixed constant). We then present a faster algorithm that takes $O(\sqrt{\log n}/\epsilon)$ rounds in undirected graphs. Both of the above algorithms are scalable, as each node processes and sends only polylog n number of bits per round.

[1]This chapter is based on joint work with Atish Das Sarma, Gopal Pandurangan and Eli Upfal and contains material from [39].

4.1 Introduction

PageRank has emerged as a very powerful measure of relative importance of nodes in a network over the last decade. The term PageRank was first introduced in [21, 83] where it was used to rank the importance of webpages on the Web. Since then, PageRank has found a wide range of applications in a variety of domains within computer science such as distributed networks, data mining, Web algorithms, and distributed computing [17, 19, 29, 67]. Since PageRank is essentially the steady state distribution corresponding to a slightly modified random walk process, it is an easily defined quantity.

While there has been work on performing random walks efficiently in distributed networks [13, 34], surprisingly, little theoretically provable results are known towards efficient distributed computation of PageRank vector. A naive way to compute PageRank of nodes in a distributed network is simply scaling iterative PageRank algorithms to distributed environment. But this is firstly not trivial, and secondly expensive even if doable. As each iteration step needs computation results of previous steps, there needs to be continuous synchronization and several messages may need to be exchanged. Further, the convergence time may also be slow. It is important to design efficient and localized distributed algorithms as communication overhead is more important than CPU and memory usage in distributed page ranking. In this chapter, we design fully decentralized algorithms for efficiently computing PageRank vector in distributed networks.

4.1.1 Outline of This Chapter

In the next Section 4.1.2, we formally state the results achieved in this chapter. Section 4.1.3 talks about the considered computing model and Section 4.1.4 defines the PageRank basics with an overview of technical approach. The first distributed algorithm for PageRank computation in general graphs and the precise results are in Section 4.2. Then a faster algorithm for PageRank computation

in undirected graphs is presented in Section 4.3. We conclude with a summary in Section 4.4.

4.1.2 Overview of Our Results

We present the first provably efficient fully decentralized algorithms for estimating PageRank vectors under a variety of settings. Our algorithms are scalable, since each node processes and sends only polylog n bits per round. Specifically, our contributions are as follows:

- We present an algorithm, Basic-PageRank-Algorithm (cf. Algorithm 7), that computes the PageRank accurately in $O(\frac{\log n}{\epsilon})$ rounds with high probability, where n is the number of nodes in the network and ϵ is the random reset probability in the PageRank random walk [10, 13, 34]. Our algorithm works for any arbitrary network i.e., directed as well as for undirected network.

- We present an improved algorithm, Improved-PageRank-Algorithm (cf. Algorithm 8), that computes the PageRank accurately in *undirected graphs* and terminates with high probability in $O(\frac{\sqrt{\log n}}{\epsilon})$ rounds. We note that though PageRank is usually applied for directed graphs (e.g., for the World Wide Web), it is sometimes also applied in connection with undirected graphs as well [4, 53, 59, 87, 91] and is non-trivial to compute (cf. Section 4.1.4). In particular, it can be applied for distributed networks when modeled as undirected graphs (as is typically the case, e.g., in P2P network models).

We note that the Improved-PageRank-Algorithm requires only $O(\log^3 n)$ bits to be sent per round per edge, whereas the Basic-PageRank-Algorithm requires only $O(\log n)$ bits per round per edge.

4.1.3 The Model of Computation

We consider almost similar distributed computing model as described in Section 1.2. The communication network is an unweighted, connected n-node graph $G = (V, E)$. Each node has limited initial knowledge. Specifically, we assume that each node is associated with a distinct identity number. At the beginning of the computation, each node v accepts as input its own identity number which is of length $O(\log n)$ bits and the identity numbers of its neighbors in G. Further, we assume a node v can communicate with any node u if v knows the id of u.[2] Initially, each node knows only the ids of its neighbors in G. We assume that the communication occurs in synchronous *rounds*, i.e., nodes run at the same processing speed and any message that is sent by some node v to its neighbors in some round r will be received by the end of round r. In each round, each node is allowed to send a message of size polylog n bits through each communication link (this applies to both communication via an edge in the network as well as direct communication). We will only focus on the running time of the algorithms.

4.1.4 PageRank Background

We formally define the PageRank of a graph $G = (V, E)$. Let ϵ be a small constant which is fixed (ϵ is called the *reset* probability, i.e., with probability ϵ, the random walk starts from a node chosen uniformly at random among all nodes in the network). The PageRank of a graph (e.g., see [10, 13, 17, 34]) is the *stationary distribution* vector π of the following special type of random walk: at each step of the walk, with probability ϵ the random walk starts from a randomly chosen node and with remaining probability $1 - \epsilon$, it follows a randomly chosen

[2]This is a typical assumption in the context of P2P and overlay networks, where a node can establish communication with another node if it knows the other node's IP address. We sometimes call this *direct* communication, especially when the two nodes are not neighbors in G. Note that our algorithm of Section 4.2 uses no direct communication between non-neighbors in G.

outgoing (neighbor) edge from the current node and moves to that neighbor.[3] Therefore the PageRank transition matrix on the state space (or vertex set) V can be written as

$$P = \left(\frac{\epsilon}{n}\right) J + (1 - \epsilon)\, Q \tag{4.1}$$

where J is the matrix with all entries 1 and Q is the transition matrix of a simple random walk on G defined as $Q_{ij} = 1/k$, if j is one of the $k > 0$ outgoing links of i, otherwise 0.

Computing the PageRank vector and its variants efficiently in various computation models has been of tremendous research interest in both academia and industry. There are mainly two broad approaches to computing PageRank (e.g., see [12]). One is to using linear algebraic techniques, (e.g., the Power Iteration [83]) and the other approach is Monte Carlo [10]. In the Monte Carlo method, the basic idea is to approximate PageRank by directly simulating the corresponding random walk and then estimating the stationary distribution with the performed walk's distribution. In [10], Avrachenkov et al. proposed the following Monte Carlo method for PageRank approximation: Perform K random walks (according to the PageRank transition probability) starting from each node v of the graph G. For each walk, terminate the walk with its first reset instead of moving to a random node. Then, the frequencies of visits of all these random walks to different nodes will approximate the PageRank. Our distributed algorithms are based on the above method.

4.2 A Distributed Algorithm for PageRank

We present a Monte Carlo based distributed algorithm for computing PageRank vector of a network [10]. The main idea of our algorithm (formal pseudocode is given in Algorithm 7) is as follows. Perform K (K will be fixed appropriately

[3] We sometime use the terminology "PageRank random walk" for this special type of random walk process.

later) random walks starting from each node of the network in parallel. In each round, each random walk independently goes to a random outgoing neighbor with probability $1-\epsilon$ and with the remaining probability (i.e., ϵ) terminates in the current node. Henceforth, we call such a random walk a *'PageRank random walk'*. In [10], this random walk process is shown to be equivalent to one based on the PageRank transition matrix P, defined in Section 4.1.4. It is easy to see that picking each node as starting point for the same number of times (i.e., restarting walks according to the uniform distribution) accounts for the $(\epsilon/n)J$ term in Equation 4.1; and between any two restarts, we just have a simple random walk that terminates with probability ϵ in each step — which accounts for the $(1-\epsilon)Q$ term. Since ϵ is the probability of termination of a walk in each round, the expected length of every walk is $1/\epsilon$ and the length will be at most $O(\log n/\epsilon)$ with high probability. Let every node v count the number of visits (say, ζ_v) of all the walks that go through it. Then, after termination of all walks in the network, each node v computes (estimates) its PageRank π_v as $\tilde{\pi}_v = \frac{\zeta_v \epsilon}{nK}$. Notice that $\frac{nK}{\epsilon}$ is the (expected) total number of visits over all nodes of all the nK walks. The above idea of counting the number of visits is a standard technique to approximate PageRank (see e.g., [10, 13]). We want to note that our algorithm in this section does not require any direct communication between non-neighbors.

We show in the next section that the above algorithm approximates PageRank vector π accurately (with high probability) for an appropriate value of K. The main technical challenge in implementing the above method is that performing many walks from each node in parallel can create a lot of congestion. Our algorithm uses a crucial idea to overcome the congestion. We show that (cf. Lemma 4.1) that there will be no congestion in the network even if we start a polynomial number of random walks from every node in parallel. The main idea is based on the Markovian (memoryless) properties of the random walks and the process that terminates the random walks. To calculate how many walks move from node i to node j, node i only needs to know the number of walks that

Algorithm 7 BASIC-PAGERANK-ALGORITHM

Input (for every node): Number of nodes n, number of walks $K = c \log n$ (where $c = \frac{2}{\delta' \epsilon}$ and δ' is defined in Section 4.2.2), reset probability ϵ.

Output: Approximate PageRank of each node.

[Each node v starts $c \log n$ walks. All walks keep moving in parallel until they terminate. The termination probability of each walk is ϵ, so the expected length of each walk is $1/\epsilon$.]

Each node v creates $c \log n$ messages, say coupons $C_1, C_2, \ldots, C_{c \log n}$.
Each node also maintains a counter ζ_v for counting the visits of random walks to it. Set $\zeta_v = 0$.
for $i = 1, 2, \ldots$ **do**
  This is i-th round. Each node v holding at least one (alive) coupon does the following in parallel:
  for each coupon C held by v (locally) **do** // [i.e., the coupons which received by v in the $(i-1)$-th round.]
    Generate a random number $r \in [0, 1]$.
    if $r < \epsilon$ **then**
      Terminate the coupon C.
    else
      Select an outgoing neighbor uniformly at random, say u. Add $T_u^v := T_u^v + 1$. // [the variable T_u^v indicates the number of coupons (or random walks) chosen to move to the neighbor u from v in the i-th round. T_u^v is reset to 0 in the beginning of every round.]
    end if
  end for
  Send the coupon counter number T_u^v to the respective outgoing neighbors u.
  Each node u computes: $\zeta_u = \zeta_u + \sum_{v \in N(u)} T_u^v$. //[the quantity $\sum_{v \in N(u)} T_u^v$ is the total number of visits of random walks to u in i-th round (from its neighbors).]
end for
Each node v outputs its PageRank as $\frac{\zeta_v \epsilon}{cn \log n}$.

reached it. It does not need to know the sources of these walks or the transitions that they took before reaching node i. Thus it is enough to send the *count* of the number of walks that pass through a node. The algorithm runs till all the walks are terminated. It is easy to see that it finishes in $O(\log n/\epsilon)$ rounds with high probability, this is because the maximum length of any walk is $O(\log n/\epsilon)$ w.h.p. Then every node v outputs PageRank as the ratio between the number of visits (denoted by ζ_v) to it and the total number of visits ($\frac{nK}{\epsilon}$) over all nodes of all the walks. We show that our algorithm computes approximate PageRanks in $O(\log n/\epsilon)$ rounds with high probability (cf. Theorem 4.2).

4.2.1 Analysis

Our algorithm computes the PageRank of each node v as $\tilde{\pi}_v = \frac{\zeta_v \epsilon}{nK}$ and we say that $\tilde{\pi}_v$ approximates original PageRank π_v. We first focus on the correctness of our approach and then analyze the running time.

4.2.2 Correctness of PageRank Approximation

The correctness of the above approximation follows directly from the main result of [10] (see Algorithm 4 and Theorem 1) and also from [13] (Theorem 1). In particular, it is mentioned in [10, 13] that the approximate PageRank value is quite good even for $K = 1$. It is easy to see that the expected value of $\tilde{\pi}_v$ is π_v (formal proof is given in [10]). Now it follows from the Theorem 1 in [13] that $\tilde{\pi}_v$ is sharply concentrated around its expectation π_v, i.e.,

$$\Pr[\mid \tilde{\pi}_v - \pi_v \mid \geq \delta \pi_v] \leq e^{-nK\pi_v\delta'} \tag{4.2}$$

where δ' is a constant depending on ϵ (the reset probability) and δ. We included the proof of the theorem below for the sake of completeness.

Theorem 4.1 (Theorem 1 in [13])**.** $\Pr[\mid \tilde{\pi}_v - \pi_v \mid \geq \delta \pi_v] \leq e^{-nK\pi_v\delta'}$

Proof. For simplicity we first show the result assuming $K = 1$. For general value of K, it will follow in the similar way. Fix an arbitrary node v. Define X_u to be ϵ times the number of visits to v in the walk started at u, Y_u to be the length of this walk, $W_u = \epsilon Y_u$, and $x_u = E[X_u]$. Then, X_u's are independent, $\tilde{\pi}_v = \frac{\sum_u X_u}{n}$ and hence $\pi_v = \frac{\sum_u x_u}{n}$, $0 \leq X_u \leq W_u$, and $E[W_u] = 1$. Then it follows easily that,

$$E[e^{tX_u}] \leq x_u E[e^{tW_u}] + 1 - x_u \leq e^{-x_u(1-E[e^{tW_u}])}$$

Thus,

$$\begin{aligned}\Pr[\tilde{\pi}_v \geq (1+\delta)\pi_v] &\leq \frac{E[e^{tn\tilde{\pi}_v}]}{e^{tn(1+\delta)\pi_v}} = \frac{\prod_u E[e^{tX_u}]}{e^{tn(1+\delta)\pi_v}} \leq \frac{\prod_u e^{-x_u(1-E[e^{tW_u}])}}{e^{tn(1+\delta)\pi_v}} \\ &= \frac{e^{-n\pi_v(1-E[e^{tW}])}}{e^{tn(1+\delta)\pi_v}} = e^{-n\pi_v(1+t(1+\delta)-E[e^{tW}])} \leq e^{-n\pi_v\delta'}\end{aligned}$$

where $W = \epsilon Y$ is a random variable with Y having geometric distribution with parameter ϵ, and $\delta' = 1 + t(1+\delta) - E[e^{tW}]$ is a constant depending on δ and ϵ, and can be found by optimization over t.

The proof for the other direction $\Pr[\tilde{\pi}_v \leq (1-\delta)\pi_v]$ is similar. □

From the above bound (cf. Equation 4.2), we see that for $K = \frac{2\log n}{\delta' n \pi_{min}}$, $\Pr[\mid \tilde{\pi}_v - \pi_v \mid \geq \delta\pi_v] \leq n^{-2}$ for any v, where π_{min} is minimal PageRank. Using union bound, it follows that there exist a node v such that $\Pr[\mid \tilde{\pi}_v - \pi_v \mid \geq \delta\pi_v]$ is at most $|V|n^{-2} = 1/n$. Hence, for all nodes v, $\mid \tilde{\pi}_v - \pi_v \mid \leq \delta\pi_v$ with probability at least $1 - 1/n$, i.e., with high probability. This implies that we get a δ-approximation of the PageRank vector with high probability for $K = \frac{2\log n}{\delta' n \pi_{min}}$. Note that δ can be arbitrary. Since the PageRank of any node is at least ϵ/n (i.e., the minimal PageRank value, $\pi_{min} \geq \epsilon/n$), so it gives $K = \frac{2\log n}{\delta'\epsilon}$. For simplicity we define that $c = \frac{2}{\delta'\epsilon}$, which is constant assuming δ (and hence δ') and ϵ are constant. Therefore, it is enough if we perform $c\log n$ PageRank random walks from each node. We note that while this value of K is sufficient to guarantee a constant approximation of the PageRanks, our algorithm permits a larger value of K, allowing for tighter approximation with the same running time (follows from Lemma 4.1 below). Now we focus on the running time of our algorithm.

4.2.3 Time Complexity

From the above section we see that our algorithm is able to compute the PageRank vector π in $O(\log n/\epsilon)$ rounds with high probability if we perform $c \log n$ walks from each node in parallel without any congestion. The lemma below guarantees that there will be no congestion even if we do a polynomial number of walks in parallel.

Lemma 4.1. *The algorithm can be implemented such that the message size is at most $O(\log n)$ per each edge in every round.*

Proof. It follows from our algorithm that each node only needs to count the number of visits of random walks to itself. Since random walks are Markovian processes, it is sufficient to send the count of random walk coupons traversing an edge in a given round. Since the total number of random walk coupons in the network is polynomially bounded, $O(\log n)$ bits suffice. Therefore, our algorithm simply sends the count, i.e., the number of coupons traversing an edge in every round. □

Theorem 4.2. *The algorithm* Basic-PageRank-Algorithm *(cf. Algorithm 7) computes a δ-approximation of the PageRanks in $O(\frac{\log n}{\epsilon})$ rounds with high probability for any constant δ.*

Proof. The algorithm outputs the RageRanks when all the walks terminate. Since the termination probability is ϵ, in expectation after $1/\epsilon$ steps, a walk terminates and with high probability (via a Chernoff bound) the walk terminates in $O(\log n/\epsilon)$ rounds. By the union bound [77], all walks (they are only polynomially many) terminate in $O(\log n/\epsilon)$ rounds with high probability. Since all the walks are moving in parallel and there is no congestion (follows from the Lemma 4.1), all the walks in the network terminate in $O(\log n/\epsilon)$ rounds with high probability. Hence the algorithm is able to output the PageRanks in $O(\log n/\epsilon)$ rounds with high probability. The correctness of the PageRanks approximation follows from [10, 13] as discussed earlier in Section 4.2.2. □

4.3 A Faster Distributed PageRank Algorithm (for Undirected Graphs)

We present a faster algorithm for PageRanks computation in *undirected* graphs. Our algorithm's time complexity holds in the bandwidth restricted communication model, requires only $O(\log^3 n)$ bits to be sent over each link in each round.

We use a similar Monte Carlo method as described in Section 4.2 to estimate PageRank. This says that the PageRank of a node v is the ratio between the number of visits of PageRank random walks to v itself and the sum of all the visits over all nodes in the network. In the previous section (cf. Section 4.2) we show that in $O(\log n/\epsilon)$ rounds, one can approximate RageRank accurately by walking in a naive way on general graph. We now outline how to speed up our previous algorithm (cf. Algorithm 7) using an idea similar to the one used in [42]. In [42], it is shown how one can perform *a* standard (simple) random walk in an undirected graph of length ℓ in $\tilde{O}(\sqrt{\ell D})$ rounds w.h.p (D is the diameter of the network). Recall that the high level idea of their algorithm is to perform 'many' short walks in parallel and later 'stitch' them to get the desired longer length walk. To apply this idea in our case, we modify our approach accordingly as speeding up (*many*) PageRank random walks is different from speeding up *one* (standard) random walk. We show that our improved algorithm (cf. Algorithm 8) approximates PageRank in $O(\frac{\sqrt{\log n}}{\epsilon})$ rounds.

4.3.1 Description of Our Algorithm

In Section 4.2, we showed that by performing $\Theta(\log n)$ walks (in particular we are performing $c \log n$ walks, where $c = \frac{2}{\delta' \epsilon}$, δ' is defined in Section 4.2.2) of length $\log n/\epsilon$ from each node, one can approximate the PageRank vector π accurately (with high probability). In this section we focus on the problem of how efficiently one can perform $\Theta(n \log n)$ walks ($\Theta(\log n)$ from each node) each of length $\log n/\epsilon$ and count the number of visits of these walks to different nodes. Throughout, by

"random walk" we mean the "PageRank random walk" (cf. Section 4.2).

The main idea of our algorithm is to first perform 'many' short random walks in parallel and then 'stitch' those short walks to get the longer walk of length $\log n/\epsilon$ and subsequently 'count' the number of visits of these random walks to different nodes. In particular, our algorithm runs in three phases. In the first phase, each node v performs $d(v)\eta$ ($d(v)$ is degree of v) independent 'short' random walks of length λ in parallel. The value of the parameters η and λ will be fixed later in the analysis. This is done naively by forwarding $d(v)\eta$ 'coupons' having the ID of v from v (for each node v) for λ steps via random walks. The intuition behind performing $d(v)\eta$ short walks is that the PageRank of an undirected graph is proportional to the degree distribution [53]. Therefore we can easily bound the number of visits of random walks to any node v (cf. Lemma 4.2). At the end of this phase, if node u has k coupons with the ID of a node v, then u is a destination of k walks starting at v. Note that just after this phase, v has no knowledge of the destinations of its own walks, but it can be known by direct communication from the destination nodes. The destination nodes (at most $d(v)\eta$) have the ID of the source node v. So they can contact the source node via *direct* communication. We show that this takes at most constant number of rounds as only polylogarithmic number of bits are sent (since η will be at most $O(\log^2 n/\epsilon)$, as shown later). It is shown that the first phase takes $O(\frac{\lambda}{\epsilon})$ rounds with high probability (cf. Lemma 4.3).

In the second phase, starting at source node s, we 'stitch' some of the λ-length walks prepared in first phase (note that we do this for every node v in parallel as we want to perform $\Theta(\log n)$ walks from each node). The algorithm starts from s and randomly picks one coupon distributed from s in Phase 1. We now discuss how to sample one such coupon randomly and go to the destination vertex of that coupon. One simple way to do this is as follows: In the end of Phase 1, each node v knows the destination node's ID of its $d(v)\eta$ short walks (or coupons). When a coupon needs to be sampled, node s chooses a random coupon number C

(from the unused set of coupons) and informs the destination node (which will be the next stitching point) holding the coupon C (by direct communication, since s knows the ID of the destination node at the end of the first phase). Let v be the destination node of C. The source s then sends a 'token' to v and s deletes the coupon C (so that C will not be sampled again next time at s, otherwise, randomness will be destroyed). The process then repeats. That is, the node v currently holding the token samples one of the coupons it distributed in Phase 1 and forwards the token to the destination of the sampled coupon, say u. Nodes v, u are called 'connectors' — they are the endpoints of the short walks that are stitched. A crucial observation is that the walk of length λ used to distribute the corresponding coupons from s to v and from v to u are independent random walks. Therefore, we can stitch them to get a random walk of length 2λ. We therefore can generate a random walk of length $3\lambda, 4\lambda, \ldots$ by repeating this process. We do this until we have completed a length of at least $((\log n/\epsilon) - \lambda)$. Then, we complete the rest of the walk by doing the naive random walk algorithm. We show that Phase 2 finishes in $O(\frac{\log n}{\lambda\epsilon} + \lambda)$ rounds (cf. Lemma 4.5).

In the third phase we count the number of visits of all the random walks to a node. As we have discussed, we have to create many short walks of length λ from each node. Some short walks may not be used to make the long walk of length $\log n/\epsilon$. We show a technique to count all the used short walks' visits to different nodes. Remember that after completion of Phase 2, all the $\Theta(n \log n)$ long walks ($\Theta(\log n)$ from each node) have been stitched. During stitching (i.e., in Phase 2), each connector node (which is also the end point of the short walk) should remember the source node of the short walk. Now start from the each connector node and do a walk in reverse direction (i.e., retrace the short walk backwards) to the source node in parallel. During the reverse walk, simply count the visits to nodes. It is easy to see that this will take at most $O(\lambda)$ rounds with high probability (cf. Lemma 4.6). Now we analyze the running time of our algorithm Improved-PageRank-Algorithm. The compact pseudo code is given below

Algorithm 8 IMPROVED-PAGERANK-ALGORITHM

Input (for every node): Number of nodes n, walks $K = c \log n$ (where $c = \frac{2}{\delta' \epsilon}$ and δ' is defined in Section 4.2.2), reset probability ϵ, short walk length $\lambda = \sqrt{\log n}$.

Output: Approximate PageRank of each node.

Phase 1: (Each node v performs $d(v)\eta = d(v) \log^2 n/\epsilon$ random walks of length $\lambda = \sqrt{\log n}$. At the end of this phase, there are $d(v) \log^2 n/\epsilon$ (not necessarily distinct) nodes holding a 'coupon' containing the ID of v.)

1: **for** each node v **do**
2: Construct $d(v) \log^2 n/\epsilon$ messages $C = \langle ID_v, \lambda \rangle$. // [messages containing nodes ID and the desired walk length of $\lambda = \sqrt{\log n}$. We will refer to these messages created by node v as 'coupons created by v'.]
3: **end for**
4: **for** $i = 1$ to λ **do**
5: This is the i-th iteration. Each node v holding at least one coupon does the following in parallel:
6: **for** each coupon C held by v **do** // [i.e., the coupons which received by v in the $(i-1)$-th iteration.]
7: Generate a random number $r \in [0, 1]$.
8: **if** $r < \epsilon$ **then**
9: Terminate the coupon C and keep the coupon as then v itself is the destination.
10: **else**
11: pick a neighbor u uniformly at random for the coupon C and forward C to u.
12: **end if**
13: **end for**
{Note that an iteration could require more than 1 round, because of congestion}
14: **end for**
15: Each destination node sends its ID to the source node, as it has the source node's ID now. // [destination nodes hold the short random walk coupon(s) C and contact the source nodes through *direct* communication.]

{algorithm continued...}

Phase 2: (Stitch short walks by token forwarding. Stitch $\Theta(\sqrt{\log n}/\epsilon)$ walks, each of length $\sqrt{\log n}$)

16: Each node v creates $K = c \log n$ messages called "tokens" which contain the ID of v as source node. // [The algorithm will forward the token around and keep track of a set of connectors (stitching points).]

17: **for** $i = 1$ to $(\sqrt{\log n}/\epsilon - 1)$ **do**

18: This is i-th round. Each node v holding at least one *token* does the following in parallel:

19: Consider each token held by v which is received in the $(i-1)$-th round. v samples one of the coupons distributed by v (in Phase 1) uniformly at random from the unused set of coupons. If the sampled coupon survives further, let u be the destination node of the coupon. Add $T_u^v := T_u^v + 1$. // [the variable T_u^v indicates the number of tokens (or stitching requests) chosen to move to u from v in the i-th round. T_u^v is reset to 0 in the beginning of every round.]
{Note that a coupon or short walk may not survive up to $\log n/\epsilon$ length.}

20: v sends the token counter number T_u^v to u and deletes all the corresponding coupons. // [this is done by *direct* communication, requires $O(\log n)$ bits.]

21: Each node u stores the count $\mathcal{U}^v = \mathcal{U}^v + \sum_v T_u^v$. //[sum up these counts T_u^v for each sender separately for backtracking in Phase 3.]

22: **end for**

23: Walk naively from the last sampled coupon's destination until all the nK tokens terminate. // [this will be at most another $\lambda = \sqrt{\log n}$ steps, since the maximum length of each walk is at most $\log n/\epsilon$ w.h.p.]

Phase 3: (Counting the number of visits of short walks to a node)

1: Each node w maintains a counter ζ_w to keep track of the number of visits of walks.

2: Each node u which stored $\mathcal{U}^v$ as stitching requests from some node v in Phase 2, does the following in parallel:

3: For all such nodes v, u uses the budget of short walks that came from this sender v (in Phase 1) and traces all the short random walks in reverse.

4: Count the number of visits to any node w during this reverse tracing and add to ζ_w. Count the visits by 'naively walking' walks.

5: Each node v outputs its PageRank π_v as $\frac{\zeta_v \epsilon}{cn \log n}$.

in Algorithm 8.

4.3.2 Analysis

First we are interested in the value of η i.e., how many coupons (short walks) do we need from each node to successfully answer all the stitching requests. Notice that it is possible that $d(v)\eta$ coupons are not enough (if η is not chosen suitably large): We might forward the token to some node v many times in Phase 2 and all coupons distributed by v in the first phase may be deleted. In other words, v is chosen as a connector node many times, and all its coupons have been exhausted. If this happens then the stitching process cannot progress. To fix this problem, we use an easy upper bound of the number of visits to any node v of a random walk of length ℓ in an undirected graph: $d(v)\ell$ times. Therefore each node v will be visited as a connector node at most $O(d(v)\ell)$ times with high probability. This implies that each node does not have to prepare too many short walks.

The following lemma bounds the number of visits to every node when we do $\Theta(\log n)$ walks from each node, each of length $\log n/\epsilon$ (note that this is the maximum length of a long walk, w.h.p).

Lemma 4.2. *If* each *node performs* $\Theta(\log n)$ *random walks of length* $\log n/\epsilon$*, then no node* v *is visited more than* $O(\frac{d(v)\log^2 n}{\epsilon})$ *times with high probability.*

Proof. We show the above bound on the number of visits still holds if each node v performs $\Theta(d(v)\log n)$ random walks of length $\log n/\epsilon$. Suppose each node v starts $\Theta(d(v)\log n)$ simple random walks in parallel. We claim that after any given number of steps i, the expected number of random walks at node v is still $\Theta(d(v)\log n)$. Consider the random walk's transition probability matrix A. Then, $A\mathbf{x} = \mathbf{x}$ holds for the stationary distribution $\mathbf{x}$ having value $\frac{d(v)}{2m}$, where m is the number of edges in the graph. Now the number of random walks started at any node v is proportional to its stationary distribution, therefore, in expectation, the number of random walks at any node after i steps remains the same. We show this

is true with high probability using Chernoff bound technique, since the random walks are independent. For each random walk coupon C, any $i = 1, 2, \ldots, \log n/\epsilon$, and any vertex v, we define $W_C^i(v)$ to be the random variable having value 1 if the random walk C is at v after i^{th} step. Let $W^i(v) = \sum_{C:\text{random walk}} W_C^i(v)$, i.e., $W^i(v)$ is the total number of random walks are at v after i^{th} step. By Chernoff bound, for any vertex v and any i,

$$\Pr[W^i(v) \geq 18d(v)\log n] \leq 2^{-3d(v)\log n} \leq n^{-3}.$$

It follows that the probability that there exists an vertex v and an integer $1 \leq i \leq \log n/\epsilon$ such that $W^i(v) \geq 18d(v)\log n$ is at most $|V(G)|(\log n/\epsilon)n^{-3} \leq \frac{1}{n}$ since $|V(G)| = n$ and $\log n/\epsilon \leq n$. Therefore, $W^i(v) \leq 18d(v)\log n$ for all v and for all i, with high probability.

Now, if each node starts $\Theta(\log n)$ independent random walks that terminate with probability ϵ in each step, the number of random walks to any node v is dominated from above by $\Theta(d(v)\log n)$. This is because there will be no more than $n\log n$ random walk coupons in the network in each step. Therefore, the total number of visits by all random walks to any node v is bounded by $O(d(v)\log^2 n/\epsilon)$ w.h.p., since there are total of $\log n/\epsilon$ steps. □

It is now clear from the above lemma (cf. Lemma 4.2) that $\eta = O(\log^2 n/\epsilon)$ i.e., each node v has to prepare $O(d(v)\log^2 n/\epsilon)$ short walks of length λ in Phase 1. Now we show the running time of algorithm (cf. Algorithm 8) using the following lemmas.

Lemma 4.3. *Phase 1 finishes in $O(\frac{\lambda}{\epsilon})$ rounds.*

Proof. It is known from Lemma 4.2 that in Phase 1, each node v performs $O(d(v)\log^2 n/\epsilon)$ walks of length λ. Assume that initially each node v starts with $d(v)\log^2 n/\epsilon$ coupons (or messages) and each coupon takes a random walk according to the PageRank transition probability. Now, in the similar way we showed in Lemma 4.2 that after any given number of steps j $(1 \leq j \leq \lambda)$,

the expected number of coupons at any node v is $d(v)\log^2 n/\epsilon$. Therefore, in expectation the number of messages, say X, that want to go through an edge in any round is at most $2\log^2 n/\epsilon$ (from the two end points of the edge). This is because the number of messages, the edge receives from its one end node, say u, in expectation is exactly the number of messages at u divided by $d(u)$. Using Chernoff bound we get, $\Pr[X \geq 24\log^2 n/\epsilon] \leq 2^{-4\log^2 n/\epsilon} \leq n^{-4}$. By union bound we get that there exists an edge and an integer $1 \leq j \leq \lambda$ such that the probability of $X \geq 24\log^2 n/\epsilon$ is at most $|E(G)|\lambda n^{-4} \leq \frac{1}{n}$, since $|E(G)| \leq n^2$ and $\lambda < n$. Hence the number of messages that go through any edge in any round is at most $24\log^2 n/\epsilon = O(\log^2 n/\epsilon)$ with high probability. So the message size will be at most $O(\log^3 n/\epsilon)$ bits w.h.p. over any edge in each round, as the source IDs (each of size $\log n$) are required to be sent. Since our considered model allows polylogarithmic (i.e., $O(\log^3 n)$) bits messages per edge per round, we can extend all the random walk's length from i to length $i+1$ in $O(1/\epsilon)$ rounds. Therefore, for walks of length λ it takes $O(\lambda/\epsilon)$ rounds as claimed. □

Lemma 4.4. *One stitching step from each node always finishes within $O(1)$ rounds.*

Proof. Each node knows all of its short walks' (or coupons') destination address. Each time when a source or connector node wants to stitch, it randomly chooses its unused coupons created in Phase 1. Then it contacts the destination nodes (holding the coupons) through *direct* communication and informs the destination nodes as the next connector node or stitching point. Note that each node only sends the count of stitching requests to the destination nodes. Therefore, $O(\log n)$ bits suffice. Since the network allows $O(\log^3 n)$ congestion, this will finish in constant rounds. □

Lemma 4.5. *Phase 2 finishes in $O(\frac{\log n}{\lambda\epsilon} + \lambda)$ rounds.*

Proof. Phase 2 is for stitching short walks of length λ to get a long walk of length $O(\log n/\epsilon)$. Therefore it needs to stitch approximately $O(\log n/\lambda\epsilon)$ times. Since

each stitch can be done in constant number of rounds (cf. Lemma 4.4) and the naive walking in parallel can take at most λ rounds, Phase 2 finishes in $O(\frac{\log n}{\lambda\epsilon}+\lambda)$ rounds. □

Lemma 4.6. *Phase 3 finishes in $O(\lambda)$ rounds.*

Proof. Each short walk is of length λ. Phase 3 is simply tracing back the short walks. So it is easy to see that we can perform all the reverse walks in parallel in $O(\lambda)$ rounds (same as the time to do all the short walks in parallel in Phase 1). Due to Lemma 4.3 and the fact that each node can communicate a $\log^3 n$ number of bits in every round, it follows that Phase 3 finishes in $O(\lambda)$ rounds. □

Now we are ready to show the main result of this section.

Theorem 4.3. *The* Improved-PageRank-Algorithm *(cf. Algorithm 8) computes PageRanks accurately with high probability and finishes in $O(\frac{\sqrt{\log n}}{\epsilon})$ rounds.*

Proof. The algorithm Improved-PageRank-Algorithm consists of three phases. We have calculated above the running time of each phase separately. Now we want to compute the overall running time of the algorithm by combining these three phases and by putting appropriate value of parameters. By summing up the running time of all the three phases, we get from Lemmas 4.3, 4.5, and 4.6 that the total time taken to finish the Improved-PageRank-Algorithm is $O(\frac{\lambda}{\epsilon} + \frac{\log n}{\lambda\epsilon} + 2\lambda)$ rounds. Choosing $\lambda = \sqrt{\log n}$, gives the required bound as $O(\frac{\sqrt{\log n}}{\epsilon})$. □

4.4 Conclusion

In this chapter, we have presented fast distributed algorithms for computing PageRank, a measure of fundamental interest in networks. Our algorithms are Monte-Carlo and based on the idea of speeding up random walks in a distributed

network. Our faster algorithms take time only sub-logarithmic in n which can be useful in large-scale, resource-constrained, distributed networks, where running time is especially crucial. Since they are based on random walks, which are lightweight, robust, and local, they can be amenable to self-organizing and dynamic networks.

Chapter 5

Distributed Sparse Cuts Computation

In this chapter, we focus on developing fast distributed algorithms for computing sparse cuts in networks[1]. Recall that, given an undirected n-node network G with conductance ϕ, the goal is to find a cut set whose conductance is close to ϕ.

We present two distributed algorithms that find a cut set with sparsity $\tilde{O}(\sqrt{\phi})$. Recall that $\tilde{O}$ hides polylog n factors. Both algorithms work in the CONGEST distributed computing model and output a cut of conductance at most $\tilde{O}(\sqrt{\phi})$ with high probability, in $\tilde{O}(\frac{1}{b}(\frac{1}{\phi}+n))$ rounds, where b is balance of the cut of given conductance. In particular, to find a sparse cut of constant balance, our algorithms take $\tilde{O}(\frac{1}{\phi}+n)$ rounds. Our algorithms can also be used to output a *local* cluster, i.e., a subset of vertices near a given source node, and whose conductance is within a quadratic factor of the best possible cluster around the specified node. Both our distributed algorithm can work without knowledge of the optimal ϕ value and hence can be used to find approximate conductance values both globally and with respect to a given source node. We also give a lower bound on the time needed for any distributed algorithm to compute any non-trivial sparse cut — any distributed approximation algorithm (for any non-trivial approximation ratio) for computing sparsest cut will take $\tilde{\Omega}(\sqrt{n}+D)$ rounds,

[1]This chapter is based on joint work with Atish Das Sarma and Gopal Pandurangan and contains material from [38].

where D is the diameter of the graph.

5.1 Introduction

Developing distributed algorithms for computing key metrics of a communication network is an important research goal with various applications. Network properties — which depend on the collective behavior of nodes and links — characterize global network performance such as routing, sampling, information dissemination, etc. These in turn depend on topological properties of the network such as high connectivity, low diameter, high conductance, and good spectral properties [49]. The above properties, all of which are critical, need to be measured periodically. Having a highly-connected network is good for fault-tolerance and reliable routing, since a packet can be routed via many disjoint paths. Low diameter ensures that packets can be routed quickly with short delay. Conductance measures how "well-knit" the network is; it determines how fast a random walk converges to the stationary distribution — known as the *mixing time*[2]. Conductance is related to the *expansion*, *spectral gap*, and mixing time of a graph. High expansion and spectral gap means that the graph has fast mixing time. Such a network supports fast random sampling which has many applications [42] and low-congestion routing [49].

In this chapter, we focus on developing fast distributed algorithms for computing sparse cuts in networks. We note that computing the minimum conductance cut — the one with conductance ϕ of the network— is NP-hard [76]. We present two distributed algorithms that find a cut set with sparsity $\tilde{O}(\sqrt{\phi})$. Our algorithms can be useful in efficiently finding sparse cuts (and their conductance values) and critical edges (the edges crossing sparse cuts) in distributed networks, which in turn can be helpful in the design, analysis, and maintenance of

[2] It is known that conductance (ϕ), mixing time (τ_{mix}), and spectral gap ($1-\lambda_2$, where λ_2 is the second largest eigenvalue of the transition matrix) are related to each other [60]: $\frac{1}{1-\lambda_2} \leq \tau_{mix} \leq \frac{\log n}{1-\lambda_2}$ and $\Theta(1-\lambda_2) \leq \phi \leq \Theta(\sqrt{1-\lambda_2})$.

topologically-(self)aware networks. In particular, the work of [49] shows how critical edges can be used to design algorithms to improve search, reduce congestion in routing, and for keeping the graph well-connected (topology maintenance). Such algorithms can be useful in the design and deployment of *reconfigurable networks* whose topology can be changed by rewiring edges such as peer-to-peer networks and wireless mesh networks. The paper [25] studied information spreading where they used a generalized notion of conductance as a key tool. In fact, the conductance helps to identify bottlenecks in the network and thus achieves fast information spreading.

Our approach, on a high-level, is based on efficiently implementing the methods of Lovász and Simonovits [4, 71, 89]. This method uses random walks to estimate the probability distribution of such walks terminating at nodes. This probability distribution can then be used to identify sparse cuts. Our first algorithm uses standard random walks (cf. Section 5.2). Our second algorithm is a variant of the first one which uses random walks with *reset* to a given source node, in other words, it computes *personalized PageRank* (cf. Section 5.4).

5.1.1 Model, Notations and Definitions

This chapter also consider the standard CONGEST model of computation. We refer Section 1.2 for detailed description. To remind briefly, we assume the communication network as an undirected, unweighted[3], connected graph $G = (V, E)$ with $|V| = n, |E| = m$. Initially, every node has limited knowledge. In every round nodes can communicate with its neighbors by sending a message of size at most $O(\log n)$. We only focus on the *running time*, i.e. the number of *rounds* of distributed algorithms. Recall that in the CONGEST model, it is rather trivial to solve a problem in $O(m)$ rounds, where m is the number of edges in the network, since the entire topology (all the edges) can be collected at one

[3]We restrict our attention to unweighted graphs for the upper bound analysis, however, our algorithm can be extended to weighted graphs as well.

node and the problem solved locally. The goal is to design faster algorithms.

We present notations that we use throughout the chapter. Let $p_\ell(s,t)$ denote the probability that a random walk of length ℓ starting from s ends in t. In fact, $p_\ell(s,t)$ is the probability distribution over the nodes after a walk of length ℓ starting from s. We simply use $p(t)$ instead of $p_\ell(s,t)$ when source node and length is clear from the text. Let S be a subset of V. We denote a partition or cut by $(S,\bar{S})$ or sometimes by $(S, V \setminus S)$ interchangeably. For a probability distribution $p(i)$ on nodes, let $\rho_p(i) = p(i)/d(i)$, where $d(i)$ is the degree of the node i. Let π_p denote the ordering of nodes in decreasing order of $\rho_p(i)$. We denote the conductance of the graph G by ϕ. The conductance and balance of a cut is formally defined below.

Definition 5.1 (**Conductance and Sparsity**). *The conductance of a cut $(S,\bar{S})$ (also called as sparsity) is $\phi(S) = \frac{|E(S,\bar{S})|}{\min\{vol(S), 2m - vol(S)\}}$ where $|E(S,\bar{S})|$ is the number of edges crossing the cut $(S,\bar{S})$ and $vol(S)$ is the sum of the degrees of nodes in S. The conductance of the graph G is $\phi(G) = \min_{S \subseteq V} \phi(S)$. We denote it by only ϕ, if it is clear from the text.*

Definition 5.2 (**Balance**). *The balance of a cut $(S,\bar{S})$ is defined as $\min\{\frac{|S|}{|V|}, \frac{|\bar{S}|}{|V|}\}$ and is denoted by b.*

Definition 5.3 (**Local Cluster**). *A local cluster with respect to a given vertex v is a subset $S \subset V$ containing v such that the conductance of $(S,\bar{S})$ is within a quadratic factor of the best possible local cluster containing v.*

5.1.2 Overview of Our Results

Our main contributions are two distributed algorithms in the CONGEST model to find sparse cuts with approximation guarantees. Both our algorithms crucially use random walks.

Theorem 5.1. *(cf. Section 5.2) Given an n-node network G with a cut of balance b and conductance at most ϕ, there is a distributed algorithm* SparseCut

(cf. Algorithm 10) that outputs a cut of conductance at most $\tilde{O}(\sqrt{\phi})$ *with high probability, in* $\tilde{O}(\frac{1}{b}(\frac{1}{\phi}+n))$ *rounds. In particular, to find a cut of constant balance, the* SparseCut *algorithm takes* $\tilde{O}(\frac{1}{\phi}+n)$ *rounds and finds a cut (if it exists) with similar approximation.*

The second algorithm is a variant of the first algorithm and based on a PageRank-based approach. The second algorithm achieves the similar running time bound as above.

Using the above results, we also show:

Theorem 5.2. *(cf. Section 5.3) Given an* n*-node network* G *and source node* s*, there is a distributed algorithm that outputs a* local *cluster in* $\tilde{O}(\frac{1}{\phi}+n)$ *rounds, where* ϕ *is the conductance of the graph.*

To prove the above running time bound, we derive a technical result on computing conductances of n (different) cuts in linear time (cf. Lemma 5.3). We note that the time bound of $\tilde{O}(\frac{1}{\phi}+n)$ is linear in n (the number of nodes) and $1/\phi$. From the definition of conductance (cf. Definition 5.1), it is clear that for every graph, $1/\phi = O(m)$ (m is the number of edges) and for many graphs it can be much smaller, e.g., for expanders it is $O(1)$. Hence, the running time of our algorithms can be significantly faster than the naive bound of $O(m)$ (cf. Section 1.2), especially in well-connected dense graphs. We next show a lower bound on the time needed for any distributed algorithm to compute a (non-trivial) sparse cut.

Theorem 5.3. *(cf. Section 5.5) There is a* n*-node graph in which any distributed approximation algorithm for computing sparsest cut (within any non-trivial approximation ratio) will take* $\tilde{\Omega}(\sqrt{n}+D)$ *rounds, where* D *is the diameter of the graph.*

Since $1/\phi = \tilde{\Omega}(D)$ for any graph [26], the above lower bound says that in general, one cannot hope to improve on the $1/\phi$ term of our upper bound.

5.1.3 Outline of This Chapter

The next two sections develop two different approaches to computing sparse cuts. In Section 5.2, we present the standard random walk-based distributed algorithm for sparse cut problem, by introducing the main ideas. Section 5.3 describes on finding local cluster set. Then, in Section 5.4 we present the second algorithm using PageRank-based approach. Section 5.5 derive a general lower bound to the sparse cut computation problem. Finally, we conclude in Section 5.6 by summarizing the results developed in this chapter and discuss some open problems.

5.2 A Distributed Algorithm for Sparse Cut

In this section, we present an algorithm to find a cut that approximates the minimum conductance ϕ. We are given a network, $G = (V, E)$, that has cut of conductance ϕ and balance b. We design a distributed algorithm running on G to compute a cut set S with conductance $\tilde{O}(\sqrt{\phi})$. At the end of the algorithm, every node in G will know whether it is in S or $\bar{S}$. Further, each node will also know all other nodes in S or $\bar{S}$. Our algorithm works in the standard CONGEST model of distributed computing. Without loss of generality, we will assume that our algorithm knows ϕ and b. Otherwise, we can do the following. Suppose we want to find a cut with the required sparsity, i.e., $\tilde{O}(\sqrt{\phi})$, without knowing ϕ, but assume that we know b (the balance of such a cut). Then we can guess the value of ϕ starting from a constant, say $1/2$, which is essentially the highest possible and then run our algorithm and check whether the output cut value satisfies the quadratic factor approximation and the given balance. If yes, we stop; otherwise, we halve our guess and continue. If we don't know b as well, then our algorithm with some assumed balance will still work and will give a cut with similar quadratic approximation to the minimum conductance cut that is minimum among all possible cuts with the assumed balance. Thus, henceforth

we will assume that our algorithm knows both ϕ and b.

The outline of our approach (cf. Theorem 5.4) is to try several different cuts obtained by various distributions of random walks. Further these distributions need to be computed from a *good* source node. A good source node is one from the smaller side of the desired cut. In this approach, instead of computing the exact distribution after the chosen length of walk, it suffices to have an approximate distribution of sufficiently high accuracy. Assuming that a good source is used, one needs to estimate the distribution after doing a random walk of length ℓ that is sampled uniformly in the range of $\{1, 2, \ldots, O(1/\phi)\}$. For the sampled length ℓ, estimate the landing probability $p(i)$ at every node i. Assume the estimation is $\tilde{p}(i)$. Then arrange the nodes according to decreasing order of $\rho_{\tilde{p}} = \tilde{p}(i)/d(i)$. Suppose the order is $\pi_{\tilde{p}} = (1, 2, \ldots, n)$. Then, with constant probability, at least one of the n cuts $(S_j, \bar{S}_j)$ has the given conductance (approximated), where $S_j = \{1, 2, \ldots, j\}$. This algorithm and its proof of correctness was given in Spielman and Teng [89]. To get the required cut with high probability, we run our algorithm for $\Theta(\log n)$ different lengths ℓ, each chosen independently and uniformly at random in the range of $\{1, 2, \ldots, O(1/\phi)\}$. For a particular ℓ, there are $(n-1)$-partitions and so $(n-1)$ different conductances. The minimum conductance cut among all the $\Theta(n \log n)$ cuts would be the output of our algorithm. Before going to the main algorithm SparseCut, we first present an algorithm to estimate the probability distribution $\tilde{p}(i)$ of $p(i)$ using random walks.

5.2.1 Estimating Random Walk Probability Distribution

We focus on estimating $p_\ell(s, i)$ which is the probability of landing at node i after a random walk of length ℓ from a specific source node s. As we noted above, we denote it by simply $p(i)$. The basic idea is to perform several random walks of length ℓ from s and at the end, each node i computes the fraction of walks that land at node i. It is easy to see that the accuracy of estimation is dependent

on the number of random walks that are performed from s. Let us parameterize the number as K. We show (cf. Lemma 5.2) that we can perform a polynomial in n number of random walks without any congestion in the network. We first present the algorithm EstimateProbability, and then describe the result on accuracy of the estimation (cf. Lemma 5.1). The pseudocode of the algorithm EstimateProbability is given below (cf. Algorithm 9).

Algorithm 9 EstimateProbability

Input: Starting node s, length ℓ, and number of walks K.

Output: $\tilde{p}(i)$ for each node i, which is an estimate of $p(i)$ with explicit bound on additive error.

1: Node s creates K tokens of random walks and performs them simultaneously for ℓ steps as follows.
2: **for** each round from 1 to ℓ **do**
3: A node holding random walk tokens, samples a random neighbor corresponding to each token and subsequently sends the appropriate *count* to each neighbor. (Note that tokens do not contain any node IDs.)
4: **end for**
5: Each node i counts the number of tokens that landed on it — let this count be η_i.
6: Each node estimates the probability $\tilde{p}(i)$ as $\frac{\eta_i}{K}$.

We show that for $K = \Theta(n^2 \log n/\epsilon^2)$, the algorithm EstimateProbability (cf. Algorithm 9) gives an estimation of $p(i)$ with accuracy $p(i) \pm \epsilon/n$ for each node i. In other words, by performing $\Theta(n^2 \log n/\epsilon^2)$ random walks, if $\tilde{p}(i)$ is an estimation for $p(i)$, then $|\tilde{p}(i) - p(i)| \leq \epsilon/n$. This follows directly from the following lemma.

Lemma 5.1. *If the probability of an event X occurring is p, then in $t = 4n^2 \log n/\epsilon^2$ trials , the fraction of times the event X occurs is $p \pm \frac{\epsilon}{n}$ with high*

probability.

Proof. The proof follows from a standard Chernoff bound:

$$\Pr\left[\frac{1}{t}\sum_{i=1}^{t} X_i < (1-\delta)p\right] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{tp} < e^{-tp\delta^2/2}$$

and

$$\Pr\left[\frac{1}{t}\sum_{i=1}^{t} X_i > (1+\delta)p\right] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{tp}.$$

Where $X_1, X_2, \ldots, X_t$ are t independent identically distributed $0-1$ random variables such that $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = (1-p)$. The right hand side of the upper tail bound further reduces to $2^{-\delta tp}$ for $\delta > 2e-1$ and for $\delta < 2e-1$, it reduces to $e^{-tp\delta^2/4}$.

Let us choose $t = 4n^2 \log n/\epsilon^2$, and $\delta = \frac{\epsilon}{pn}$. Consider two cases, when $pn \leq \epsilon$ and when $pn > \epsilon$. When $pn \leq \epsilon$, the lower tail bound automatically holds as $pn - \epsilon < 0$. In this case, $\delta > 1$, so we consider the weaker bound of the upper tail bound which is $2^{-\delta tp}$. We get $2^{-\delta tp} = 2^{-\epsilon t/n} = 2^{-4n\log n/\epsilon} = \frac{1}{n^{(4n/\epsilon)}}$. Now consider the case when $pn > \epsilon$. Here, $\delta < 1$ is small and hence the lower and upper tail bounds are $e^{-tp\delta^2/2}$ and $e^{-tp\delta^2/4}$. Therefore, between these two, we go with the weaker bound of $e^{-tp\delta^2/4} = e^{-\frac{tp\epsilon^2}{4p^2n^2}} = e^{-\frac{1}{p}\log n} = 1/n^{\Theta(1)}$. □

Lemma 5.2. *Algorithm* EstimateProbability *(cf. Algorithm 9) finishes in $O(\ell)$ rounds, if the number of walks K is at most polynomial in n.*

Proof. To prove this, we first show that there is no congestion in the network if we perform at most a polynomial number of random walks from s. This follows from the algorithm that each node only needs to count the number of random walk tokens that end on it. Therefore nodes do not need to know from which source node or rather from where it receives the random walk tokens. Hence it is not needed to send the ID of the source node with the token. Since we consider CONGEST model, a polynomial in n number of token's count (i.e., we can send count of up to a polynomial number) can be sent in one message through each

edge without any congestion. Therefore, one round is enough to perform one step of random walk for all K walks in parallel, where K is at most polynomial in n. This implies that K random walks of length ℓ can be performed in $O(\ell)$ rounds. Hence the lemma. □

5.2.2 Computation of Sparse Cut

With the probability approximation result (cf. Lemma 5.1) and results from the algorithm NIBBLE in [89], a key technical result follows (stated below). The result guarantees that one of the cuts formed by n-prefixes in a specific sorted order of the probability distribution $\tilde{p}(i)$ has sparsity $\tilde{O}(\sqrt{\phi})$ [72, 89].

Theorem 5.4. *Let $(U, \bar{U})$ be a cut of conductance at most ϕ such that $|U| \leq |V|/2$. Let $\tilde{p}(i)$ be an estimate for the probability $p(i)$ of a random walk of length ℓ from a source node s from U. Assume that $|\tilde{p}(i) - p(i)| \leq \epsilon(\sqrt{p(i)/n} + 1/n)$, where $\epsilon \leq o(\phi)$. Consider the $n-1$ candidate cuts obtained by ordering the vertices in decreasing order of $\rho_{\tilde{p}}$; each candidate cut $(S_j, \bar{S}_j)$ is obtained by setting S_j equal to the set $(1, 2, \ldots, j)$. If the source node is randomly chosen from U and the length is chosen randomly in the range $\{1, 2, \ldots, O(1/\phi)\}$, then with constant probability, one of these $n-1$ candidate cuts has conductance at most $\tilde{O}(\sqrt{\phi})$, i.e. $\phi(S_j) \leq \tilde{O}(\sqrt{\phi})$.*

Proof. The proof is shown in [33] and is implicit in [89] and uses a random walk mixing result from [72]. □

Therefore, it follows from the above Theorem 5.4 that if we can estimate the probability $p(i)$ in such a way that it satisfies all the conditions as stated, then we can find a cut with sparsity $\tilde{O}(\sqrt{\phi})$. We see that the algorithm ESTIMATEPROBABILITY estimates $p(i)$ and the error bound is given in Lemma 5.1. By setting ϵ appropriately (which is $O(\phi^2)$), we can satisfy the requirement of Theorem 5.4. We only need to choose the source node s, a bit carefully. The source node s should be sampled from the smaller side of the cut of given conductance ϕ as

it is required in Theorem 5.4. But we do not have any idea about the cut. To overcome this, we sample several source nodes from V and execute the algorithm for every source node. By sampling $\log n/b$ random nodes from V gives at least one node from the smaller side of the cut with high probability. Notice that if b is constant then it is enough to choose $O(\log n)$ source nodes.

In the following lemma, we show that one can compute the conductances of n cuts, obtained according to some ordering of vertices, in linear time. In particular, we use the ordering of the vertices in decreasing order of $\rho_{\tilde{p}}$.

Lemma 5.3 ($n-1$**-Cuts' Conductance**). *Let $G=(V,E)$ be an undirected graph. Let $\pi=(1,2,\ldots,n)$ be an ordering of n vertices of G. Then computing conductances of all $n-1$ cuts $(S_j,\bar{S}_j), j=1,2,\ldots,n-1$, can be done in $O(n)$ rounds where $S_j=\{1,2,\ldots,j\}$.*

Proof. Let us assume that each node in the graph knows the ordering π, i.e., each node knows its position in the ordering π. We know from definition of conductance (cf. Definition 5.1) that only two values are needed, namely $|E(S,\bar{S})|$ (number of crossing edges between S and $\bar{S}$) and $vol(S)$ (sum of degrees of nodes in S) to compute the conductance of a cut $(S,\bar{S})$. Therefore, our goal is to collect these two pieces of *information* of all the cuts at node 1 and compute conductances locally. We assume that node 1 knows m, the number of edges in the graph, otherwise, it can be known easily using $O(D)$ rounds by building a breadth-first tree (e.g., after leader election). Notice that the partitions $(S_j,\bar{S}_j)$ are formed by adding nodes one by one from the ordered set $\{1,2,\ldots,n\}$ starting from the set $S_1=\{1\}$. Suppose node 1 has the information of L_j^π = number of neighbors of node j in S_{j-1} and R_j^π = number of neighbors of node j in $\bar{S}_j$ (assuming S_0 =NULL) for all nodes $j=1,2,\ldots,n$ (cf. Figure 5.1). Then, node 1 can easily compute the value of $|E(S_j,\bar{S}_j)|$ and $vol(S_j)$ for all partitions locally as follows: $|E(S_j,\bar{S}_j)| = |E(S_{j-1},\bar{S}_{j-1})| - L_j^\pi + R_j^\pi$ and $\mathrm{vol}(S_j) = \mathrm{vol}(S_{j-1}) + L_j^\pi + R_j^\pi$ and $|E(S_1,\bar{S}_1)| = \deg(1)$ and $\mathrm{vol}(1) = \deg(1)$ (where $deg(1)$ is node 1's degree). Therefore, node 1 starts computing from S_1, S_2, and so on up to S_n. Note that

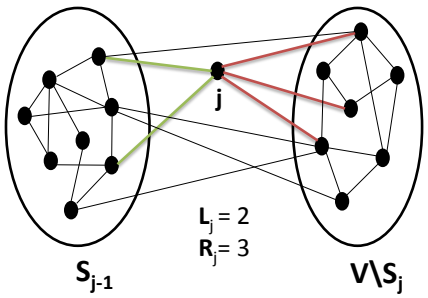


Figure 5.1: Node j computes the number of its neighbors that are in the left side and right side of j in the ordered vertex set π.

$L_j^\pi + R_j^\pi$ = degree of the node j. We next mention how node 1 can have all these information in linear time. This is easy: a node can compute its neighbor's position (i.e., whether its neighbor is in S_{j-1} or in $\bar{S}_j$) in the ordered set π in constant number of rounds (cf. Figure 5.1). Every node can do this computation in parallel. It will take constant number of rounds for every node to compute L_j^π and R_j^π. Then each node j sends the information which contains its ID, L_j^π and R_j^π to node 1 by upcast [86]. This will take at most $O(n + D)$ rounds. Then node 1 can compute conductances locally as discussed above. Therefore, total time taken is $O(n + D)$ rounds to compute all n conductances. This is actually $O(n)$ rounds, since $D \leq n$. □

The algorithm and description of each step is given below. Complete pseudocode is given in Algorithm 10 which computes an approximate cut. At the end of our algorithm, each node knows the cut set which has sparsity $\tilde{O}(\sqrt{\phi})$.

5.2.3 Description and Analysis of the Algorithm

We describe the algorithm SparseCut in detail here. First we want to compute the probability distribution of random walks starting from a source node. It is shown in [89] (cf. Theorem 5.4) that the source node should be from the smaller side of the cut of given conductance ϕ. Since we do not know about the cut set,

Algorithm 10 SparseCut

Input: Graph $G = (V, E)$, a conductance ϕ of a cut and balance b of the cut (as mentioned in the beginning of Section 5.2, we assume knowledge of ϕ and b, without loss of generality).

Output: A sparse cut $C = (S, \bar{S})$ with conductance at most $\tilde{O}(\sqrt{\phi})$.

1: **for** $k = 1$ to $\log n/b$ **do**

2: Choose a source node s_k uniformly at random from V.

3: **for** $h = 1$ to $\log n$ **do**

4: Choose a length ℓ uniformly at random in the range of $\{1, 2, \ldots, O(1/\phi)\}$.

{Phase 1: Finding partition of nodes using probability distribution by random walks.}

5: Source node s_k calls algorithm EstimateProbability with input ℓ and $K = \Theta(\frac{n^2 \log n}{\epsilon^2})$ to compute $p(i)$ for all nodes i.

6: Each node sends the value $\rho(i) = \tilde{p}(i)/d(i)$ to all other nodes in the network.

7: Let (without loss of generality) $\pi_{\tilde{p}} = \{1, 2, \ldots, n\}$ be the ordering of nodes in decreasing order of the set $\{\rho(i) : i \in V\}$. Each node knows $\pi_{\tilde{p}}$.

{Phase 2: Finding conductance of the cuts $(S_j, \bar{S}_j)$ **where** $S_j = \{1, 2, \ldots, j\}$ **for** $j = 1, 2, \ldots, n-1$ **in** $\pi_{\tilde{p}}$**.}**

8: Consider node 1 as master node which collects information of all $n-1$ cuts $(S_j, \bar{S}_j)$ one by one and computes conductances locally as follows.

9: Every node j does in parallel: compute L_j^π = number of neighbors in S_{j-1} and R_j^π = number of neighbors in $\bar{S}_j$ (assuming S_0 =NULL).

10: Each node j sends the *information* which contains their ID, L_j^π and R_j^π, to node 1.

11: Node 1 computes all the conductance of all $n-1$ cuts locally using this information.

12: $C^\ell \leftarrow$ cut of minimum conductance i.e., $\phi(C^\ell) = \min_{j=1,2,\ldots,n-1} \phi(S_j)$.

13: **end for**

{algorithm continued...}

14: Node 1 chooses the cut C_{s_k} of minimum conductance among all C^ℓ i.e., $\phi(C_{s_k}) = \min_{\ell=1}^{\log n} \phi(C^\ell)$.

15: **end for**

16: Node 1 broadcasts the cut which has minimum conductance among all C_{s_k}, to all the nodes in the network.

we cannot choose such a source node. However, if the balance of the cut is b, if we choose $\log n/b$ source nodes uniformly at random from V, then with high probability at least one node should be from the smaller side of the cut.

For each source node, we compute landing probability distribution of random walks of length ℓ. The length could be at most $O(1/\phi)$ (cf. Theorem 5.4) in the range of $\{1, 2, \ldots, O(1/\phi)\}$. As mentioned earlier, we run our algorithm for $\log n$ different lengths ℓ, chosen uniformly at random in this range. For simplicity, we break the remaining portion of the algorithm into two parts: Phase 1 and Phase 2. We run these two phases for each length ℓ and for every (chosen) source node. In Phase 1, we partition the vertex set V according to the prefixes of decreasing order of the ratio of node probability to its degree. First, source node calls the algorithm EstimateProbability with input ℓ and K to estimate landing probability over nodes. After computing approximate probability distribution $\tilde{p}(i)$, each node sends the value $\rho(i) = \tilde{p}(i)/d(i)$ to all other nodes in the network (cf. proof of the Lemma 5.4). Then, we arrange the set of vertices in decreasing order of $\rho(i)$, say, the ordered set is $\pi_{\tilde{p}} = \{1, 2, \ldots, n\}$. At the end of this phase, each node knows all the partitions $(S_j, \bar{S}_j)$, for all $j = 1, 2, \ldots, n-1$. In phase 2, we compute the conductances of these $n-1$ partitions. We describe in Lemma 5.3 on how to compute conductances of all these cuts in linear time.

There are $n-1$ partitions (cut sets) corresponding to each ℓ. Then node 1 chooses the minimum of the $\Theta(n \log n)$ (among all ℓ) minimum conductance cuts. Say, the cut is C_{s_k}, where $\phi(C_{s_k}) = \min_\ell\{\min_j \phi(S_j)\}$. Then node 1 chooses the

minimum conductance cut among all source nodes s_k (there are total $\log n/b$) and broadcasts it to all the nodes in the network. Let the minimum cut be $C = (C_t, \bar{C}_t)$, then it is enough for node 1 to broadcast the node t only as all nodes know the ordered set $\pi_{\tilde{p}}$.

5.2.4 Time Complexity Analysis

We now analyze the running time of the algorithm SparseCut. The following lemmas are required to prove the time complexity of SparseCut.

Lemma 5.4. *Phase 1 of* SparseCut *(cf. Algorithm 10) takes* $O(\frac{1}{\phi} + n)$ *rounds.*

Proof. In phase 1, we estimate probability distribution $p(i)$ using the EstimateProbability Algorithm. The running time of EstimateProbability, following Lemma 5.2, is $O(\ell)$ rounds. After estimating the landing probability, each vertex sends the quantity $\rho(i) = \tilde{p}(i)/d(i)$ to all vertices in the network. A simple way of sending these n value to n nodes can be done by constructing a BFS tree (e.g., by first electing a leader). We first construct a BFS tree using the value $\rho(t)$ of each node as its rank. Then the node of highest ρ value would be the root of the tree. Each node upcasts its ρ value to the root node through tree edges. Then the root node floods all $\rho(t)$ to reach all the nodes through the tree edges. It is shown in [86] that the upcast and then flooding n values through tree edges can all be done in $O(n + D)$ rounds, where D is the diameter of the graph. Also constructing BFS can be done in $O(D)$ rounds (e.g., [63]).

All other computations are done locally. Therefore, the total time required for Phase 1 is $O(\ell + n + D)$ rounds. However, the algorithm EstimateProbability is called for $\Theta(\log n)$ different random walk lengths, where each length value is at most $O(1/\phi)$. Also the diameter D is at most $O(n)$ for any graph. Therefore, phase 1 finishes in $O(1/\phi + n)$ rounds. □

Lemma 5.5. *Phase 2 of* SparseCut *(cf. Algorithm 10) takes* $O(n)$ *rounds.*

Proof. Phase 2 is for computing conductance of $(n-1)$ cuts $(S_j, \bar{S}_j), j = 1, 2, \ldots, n-1$ where $S_j = \{1, 2, \ldots, j\}$ according to the ordering in $\pi_{\tilde{p}}$. Therefore, it follows from the proof of Lemma 5.3 that node 1 can compute these $(n-1)$ conductances in $O(n)$ rounds. □

Theorem 5.5. *The running time of* SparseCut *(cf. Algorithm 10) is* $O(\frac{1}{b}(\frac{1}{\phi} + n)\log^2 n)$ *rounds where* ϕ *is the conductance of the graph and* b *is balance of the cut.*

Proof. The algorithm SparseCut essentially runs in two phases inside first two **for** loops, one is for choosing source nodes and other is for choosing length of random walks. Then at the end, node 1 performs some local computation to choose the minimum conductance cut and sends it to all other nodes. Sending this to all the nodes in the network can be done in $O(D)$ rounds, which follows from the above discussion of the algorithm. Now, for phase 1 and phase 2, we already have calculated the running time (cf. Lemma 5.4 and Lemma 5.5). Therefore, adding all these time together, we get $O(\frac{\log n}{b}(1/\phi + n)\log n + D)$ rounds, where the factor $\frac{\log n}{b}$ is for the first **for** loop, the factor $\log n$ for the second **for** loop and last D is for sending the cut information to all nodes. All other computations are dominated by this bound. Since D is dominated by n, therefore the running time of the algorithm SparseCut reduces to $O(\frac{1}{b}(1/\phi + n)\log^2 n)$ rounds. □

Combining the above running time lemmas, we prove the main result of this section — Theorem 5.1 (cf. Section 5.1.2) restated below.

Theorem 5.6. *Given an* n*-node network* G *with a cut of balance* b *and conductance at most* ϕ*, there is a distributed algorithm* SparseCut *(cf. Algorithm 10) that outputs a cut of conductance at most* $\tilde{O}(\sqrt{\phi})$ *with high probability, in* $\tilde{O}(\frac{1}{b}(\frac{1}{\phi} + n))$ *rounds. In particular, to find a cut of constant balance, the* SparseCut *algorithm takes* $\tilde{O}(\frac{1}{\phi} + n)$ *rounds and finds a cut (if it exists) with similar approximation.*

Proof. The approximation guarantee of Algorithm SparseCut, i.e., it computes a cut with sparsity $\tilde{O}(\sqrt{\phi})$ follows from Theorem 5.4. We choose $\epsilon = O(\phi^2)$. Moreover, we are performing random walks up to length $O(1/\phi)$. Therefore, it follows from Theorem 5.4 that our algorithm computes a cut with conductance $\tilde{O}(\sqrt{\phi})$. The running time of the algorithm follows from the above Theorem 5.5 which is $\tilde{O}(\frac{1}{b}(\frac{1}{\phi} + n))$ rounds.

In the SparseCut algorithm, we are required to compute probability distributions by performing random walks from a *good* source node to satisfy the condition of Theorem 5.4. A source node is *good* if it is from the smaller side of a desired cut (as shown in [89]). If we are interested in finding a cut of constant balance, then b is constant. Therefore, as an immediate corollary, computing a sparse cut of constant balance takes $\tilde{O}(\frac{1}{\phi} + n)$ rounds. □

The analysis of our algorithm is tight. Consider the barbell graph B_n which is a graph consisting of two cliques of size $(n-1)/2$ connected by a path of length 2 (see, figure 2 in [3]). Consider a source node s in one clique. Then to compute the smallest conductance cut (one set of which would be the clique containing s), the random walk starting from s, should reach the second clique. This will take at least $\Theta(n^2)$ rounds, which is bounded by $\Omega(1/\phi)$. Then to collect all the information as in Lemma 5.3 at the node s will take $\Omega(n)$ rounds. Hence, total time required is $\Omega(1/\phi + n)$.

5.3 Finding Local Cluster Set

We describe an approach to compute a local cluster, i.e., a subset of vertices containing a given source node v such that the internal edge connections are significantly higher than the outgoing edges from it.

Suppose a source node $s \in V$ is given. First, guess a conductance ϕ starting from a constant (say 1/2, which is essentially the best possible) and then run the above SparseCut algorithm for the particular node s, i.e., run the algorithm

from Step 3 for source node s. Then check whether the smallest conductance satisfies the quadratic factor approximation. If yes, we stop; otherwise, we halve the (guessed) conductance and continue. Since the minimum conductance value is $O(1/m)$, we need to do at most $O(\log n)$ guesses, as $m = O(n^2)$. The running time bound of the algorithm for computing a local cluster is stated in Theorem 5.2 (cf. Section 5.1.2). The proof is given below with restatement of the theorem.

Theorem 5.7. *Given an n-node network G and source node s, there is a distributed algorithm that outputs a* local *cluster in $\tilde{O}(\frac{1}{\phi} + n)$ rounds, where ϕ is the conductance of the graph.*

Proof. We run the SparseCut algorithm only for one specified source node. The running time of SparseCut algorithm for a single source node is $\tilde{O}(1/\phi + n)$ rounds with high probability. Checking whether the smallest conductance satisfies the quadratic factor approximation can be done locally at the source node s. Then we may have to run the algorithm at most $O(\log n)$ times for guessing the (best possible) conductance. Therefore, the running time of the algorithm is $\tilde{O}(1/\phi+n)$ rounds with high probability. □

5.4 Sparse Cuts using PageRank

In this section, we present another approach to compute a sparse cut of an undirected graph $G = (V, E)$. This is a variant of the first algorithm and based on PageRank computation. We derive an algorithm following [4] and adapt it to the CONGEST distributed computing model and obtain similar guarantees as before, i.e., a quadratic approximation.

Recall that in the previous section we use random walk probability distributions to find candidate partitions of the vertex set. Now instead of standard random walk, we use another well known distribution vector called *PageRank* to partition vertices. The PageRank of a graph (e.g., [21, 83]) is the *stationary distribution* vector of the following special type of random walk: at each step of

the walk, with some probability α it starts from a randomly chosen node and with remaining probability $1-\alpha$, it follows a randomly chosen neighbor from the current node and moves to that neighbor. The parameter α is called *reset* probability or *teleport* probability. The personalized PageRank (e.g., [13] and references therein) is the stationary distribution vector of a slightly modified random walk as of PageRank: In every round, instead of starting from a randomly chosen node with probability α, the walk restarts from the source node itself and with remaining probability $1-\alpha$, the walk moves to a random neighbor from the current node. This alternative approach of graph partitioning, based on personalized PageRank vectors, was studied by Andersen et al. in [4] in centralized setting. They show an improved result similar to Spielman et al. [89] using personalized PageRank vectors with better approximation and running time. In this chapter, we build on the results of [4] and present a distributed algorithm to compute sparse cuts. Along the way, we also present a simple and efficient distributed algorithm to compute personalized PageRank. Throughout this section, by random walk we mean this special type of random walk unless otherwise stated.

We next discuss estimation of personalized PageRank vectors in the distributed CONGEST model. First we introduce some notation. Let $\mathbf{p}(q)$ denote the PageRank vector with respect to a given *starting* vector q, i.e., the starting node is chosen with distribution q. The personalized PageRank vector with respect to a given node v can be denoted by $\mathbf{p}(\chi_v)$, where the starting vector χ_v is the characteristic vector of v (i.e., it is 1 at v's coordinate and 0 elsewhere). We compute an ϵ-approximate PageRank vector $\tilde{\mathbf{p}}(\chi_v)$ which is within an additive error of ϵ. For technical reasons (cf. Section 5.4.2), we take ϵ be $O(1/n^4)$.

5.4.1 Estimating Personalized PageRank

We derive a simple approach to estimate the personalized PageRank vector. We present a Monte Carlo based distributed algorithm for computing personalized PageRank of a graph, similar to [39]. The main idea is as follows. Perform many

random walks starting from a specific source node s. In every round, each random walk independently goes to a random neighbor with probability $1-\alpha$ and with the remaining probability (i.e., α) terminates in the current node. We note that the random walk here means the personalized PageRank random walk. Since, α is the probability of termination of a walk in each round, the expected length of every walk is $1/\alpha$ and the length will be at most $O(\log n/\alpha)$ with high probability. During this process, every node v counts the number of visits (say, η_v) of all the walks that go through it. Suppose the number of random walks starting from s is K. Then, after termination of all walks in this process, each node v computes (estimates) its personalized PageRank $p(v)$ as $\tilde{p}(v) = \frac{\eta_v \alpha}{K}$. Notice that $\frac{K}{\alpha}$ is the (expected) total number of visits over all n nodes of all the K walks. The above idea of counting the number of visits is a standard technique to approximate PageRank (see e.g., [10, 13]). We first present the algorithm in a pseudocode (cf. Algorithm 11) to approximate $p(v)$ and then analyze the result on accuracy of estimation below.

Analysis

Now we show that the algorithm EstimatePageRank (cf. Algorithm 11) gives an estimation $\tilde{p}(v)$ of $p(v)$ with very high accuracy. The algorithm outputs the personalized PageRank of each node v as $\tilde{p}(v) = \frac{\eta_v \alpha}{K}$. The correctness of the above approximation follows directly from the analysis of the Algorithm 1 in [39]. However, the algorithm of [39] is for computing the general PageRank (not personalized) (using an approach due to [10]). However, it is easy to verify that the approach is equivalent for both general PageRank and personalized PageRank. This is because in general PageRank computation [39], several random walks are performed from every node and the walks are terminated with reset probability (instead of restarting from a random node). Now for personalized PageRank, we perform several random walks from a *particular source node* and terminate each walk with reset probability (instead of restarting from the source node

Algorithm 11 EstimatePageRank

Input: Source node s, reset probability α, and number of walks K.

Output: Approximate PageRank $\tilde{p}(v)$ of each node v.

1: Node s floods the value $K = n^4 \log n$, the number of random walks to be performed to all other nodes.
2: Source node s creates K random walk tokens and performs these simultaneously. All walks keep moving in parallel until they TERMINATE.
3: Every node maintains a counter number η_v for counting visits of random walks to it.
4: **while** there is at least one (alive) token **do**
5: This is i-th round. Each node v holding at least one token does the following: Consider each random walk token $\mathcal{C}$ held by v which is received in the $(i-1)$-th round. Generate a random number $r \in [0, 1]$.
6: **if** $r < \alpha$ **then**
7: Terminate the token $\mathcal{C}$.
8: **else**
9: Select an outgoing neighbor uniformly at random, say u. Add one token counter number to T_u^v where the variable T_u^v indicates the number of tokens (or random walks) chosen to move to the neighbor u from v in the i-th round.
10: **end if**
11: Send the token's counter number T_u^v to the respective outgoing neighbor u.
12: Every node u adds the total counter number ($\sum_{v \in N(u)} T_u^v$—which is the total number of visits of random walks to u in i-th round) to η_u.
13: **end while**
14: Each node outputs its personalized PageRank as $\frac{\eta_v \alpha}{K}$.

again). Therefore, in both cases, the random walks start again independently with probability α from source node(s). Hence, our approach also correctly outputs the personalized PageRank vector.

It is shown in [39] that by performing total $\Theta(\log n)$ random walks from each node, we get a sharp approximation of PageRank vector with high probability. Therefore, for personalized PageRank, it is enough to get a good accuracy, if we perform $K = \Theta(n \log n)$ random walks from a particular source node. However, we can perform much more walks to get very high accuracy as needed here. In particular, we show later that it would be sufficient for our algorithm to perform $O(n^4 \log n)$ random walks. Below is a lemma on the running time of our algorithm.

Lemma 5.6. *Algorithm* EstimatePageRank *(cf. Algorithm 11) computes personalized PageRank in $\tilde{O}(\frac{1}{\alpha})$ rounds with high probability, where α is the reset probability.*

Proof. To prove the lemma, we first show that there is no congestion in the network if the source node starts at most a polynomial (in n) number of random walks simultaneously. This is because, nodes are only sending the 'count' number of random walk tokens in the algorithm. The process is similar to that in Section 5.2.1 where we estimate the landing probability distribution using the same technique. Hence the claim on the congestion part follows from the proof of the Lemma 5.2.

Now it is clear that the algorithm stops when all the walks terminate. Since the termination probability is α, so in expectation after $1/\alpha$ steps, a walk terminates and with high probability (via the Chernoff bound) the walk terminates in $O(\log n/\alpha)$ rounds; by union bound [77], all walks (since they are only polynomially many) terminate in $O(\log n/\alpha)$ rounds with high probability as well. Since all the walks are moving in parallel and there is no congestion, all the walks in the graph terminate in $O(\log n/\alpha)$ rounds w.h.p. □

5.4.2 Algorithm for Sparse Cut using PageRank

We describe an algorithm to compute a sparse cut in G. The idea is very similar to the previous section (cf. Algorithm 10). In the previous section we

used standard random walk to find the partitions of vertex set. Here we use personalized PageRank for partitioning and arrange the vertices in decreasing order of the ratio: (PageRank)/(degree of vertex). Consider $(n-1)$ partitions according to this ordering and compute conductance for each of them. Then the cut of minimum conductance is at most $\tilde{O}(\sqrt{\alpha})$, if we performed random walk from a specified vertex with reset probability $O(\alpha)$ (we will take α to be $\Theta(\phi)$). This guarantee follows from the result of [4] stated below (in modified form for our purposes).

Theorem 5.8 ([4])**.** *Let C_v be a cut containing node v with a conductance ϕ. If $\tilde{p}$ is an ϵ-approximation to the personalized PageRank vector $\mathbf{p}(\chi_v)$* [4] *computed with the reset probability $\alpha = 10\phi$, and $\epsilon = O(\frac{1}{n^4})$, then the smallest conductance of $n-1$ cuts according to the ordering of vertices in decreasing order of $\tilde{p}(i)/d(i)$ is $\phi(\tilde{p}) = \tilde{O}(\sqrt{\phi})$.*

Algorithm: Similar to SparseCut algorithm in Section 5.2. First we have to choose a *good* source node, i.e., a node s from the smaller side of the cut of a given conductance ϕ. For this, we choose $O(\log n/b)$ uniformly random nodes, assuming the balance b of the cut is given. This will guarantee that at least one node is from the smaller side with high probability. Then do the following for every source node s: compute the personalized PageRank vector using algorithm EstimatePagerank with source node s and reset probability $\alpha = 10\phi$. Compute conductances of $(n-1)$ cuts derived from the PageRank vector as explained above. Then output the cut set with minimum conductance among all $(n-1)\cdot\frac{\log n}{b}$ cuts. Notice that there are $(n-1)$ cuts for one source node. Then the output cut would have sparsity $\tilde{O}(\sqrt{\phi})$ which follows from the

[4] Actually, the result holds for a slightly different type of approximate PageRank vector defined in [4]; nevertheless, this can be shown to be closely approximated by ϵ-approximate PageRank vector $\tilde{p}$ as defined here, if we choose ϵ small enough, i.e., $\epsilon = O(1/\mathrm{vol}(C_v)^2)$, i.e., $O(1/n^4)$.

Theorem 5.8. The reset probability α of the PageRank is chosen 10ϕ according to the theorem 5.8. The following theorem state the main result of this section.

Theorem 5.9. *Given any graph G and a conductance at most ϕ, there is a PageRank based algorithm that computes a cut set of conductance at most $\tilde{O}(\sqrt{\phi})$ with high probability in $\tilde{O}(\frac{1}{b}(\frac{1}{\phi}+n))$ rounds, where b is the balance of the cut.*

Proof. The algorithm runs in two phases for each of $O(\log n/b)$ source nodes. The first phase is for computing personalized PageRank and it takes $\tilde{O}(\frac{\log n}{\phi})$ rounds with high probability (cf. Lemma 5.6). The second phase is similar to the Algorithm 10. That is, computing partitions according to the PageRank, and then computing conductances of all partitions: all this can be done in $\tilde{O}(n+D)$ rounds. Hence totally we have $\tilde{O}(1/\phi+n)$ rounds with high probability, since diameter $D \leq n$. Therefore, over all the $O(\log n/b)$ source nodes, the running time of the PageRank based algorithm is $\tilde{O}(\frac{1}{b}(\frac{1}{\phi}+n))$ rounds with high probability. □

5.5 Lower Bound

We derive a general lower bound for the distributed sparse cut problem. In particular, we show that there is graph in which any approximation algorithm for computing sparsest cut will take $\tilde{\Omega}(\sqrt{n}+D)$ rounds, where D is the diameter of the graph. We use the technique of [35] which shows almost tight lower bounds for many distributed verification and optimization problems. Their lower bound proofs rely on a bridge between communication complexity and distributed computing.

We show a reduction from the spanning connected subgraph *verification problem* to the sparsest cut (optimization) problem. In the spanning connected subgraph verification problem, given a graph $G = (V, E)$ and a subgraph $H = (V, E')$ with $E' \subseteq E$, it is required to check whether the subgraph H is a spanning connected subgraph of G via a distributed algorithm. We convert the spanning connected subgraph verification problem to the sparsest cut problem (with edge

weights). In particular, we show that an α-approximation ϵ-error algorithm[5] $\mathcal{A}$ for sparsest cut problem, can be used to solve the spanning connected subgraph verification problem using the same running time. Hence the lower bound proved in [35] (cf. Theorem 5.1) for the spanning connected subgraph verification problem (which is $\tilde{\Omega}(\sqrt{n} + D)$), also applies to the sparsest cut computation problem. We use the graph $G(\Gamma, d, p)$ (this graph with parameters Γ, d, and p is defined in [35]) to show the lower bounds. We consider the same parametrized graph $G = G(\Gamma, d, p)$, which is connected by our assumption. The reduction from the spanning connected subgraph verification problem is direct: In G we assign a weight of 1 to all edges in the subgraph H and weight 0 to all other edges. Now, observe that if H is not connected then the conductance of sparsest cut is 0, since we can then partition the whole graph into two components and all the edges crossing the two components has weight 0. On the other hand, if H is connected then every cut set contains at least one edge from H, which implies that the conductance of the sparsest cut would be non-zero. Thus, any algorithm with non-trivial approximation ratio will be able to distinguish the two cases.

Therefore, it follows that the sparsest cut computation problem has a lower bound $\tilde{\Omega}(\sqrt{n} + D)$.

5.6 Conclusion

We presented distributed approximation algorithms for computing sparse cuts, with provable guarantees on the conductance. For future work, one may try to improve the running time bound $\tilde{O}(\frac{1}{\phi} + n)$ rounds. The work on performing an ℓ length random walk in time $\tilde{O}(\sqrt{\ell D})$ rounds [42], can be used to potentially speed up random walks and hence reduce the "$\frac{1}{\phi}$ part" of the time bound, since walks of that much length has to be performed. However, the technique in [42]

[5]A randomized algorithm $\mathcal{A}$ is α-approximation ϵ-error if for any input , the algorithm $\mathcal{A}$ outputs a solution that is at most α times the optimal solution of the input with probability at least $1 - \epsilon$.

may not be applicable directly here because of congestion; we need to perform many random walks to compute the landing probability distribution with high enough accuracy. One might also try to improve the "n" part of the time bound; however improving this seems to depend on computing the conductance of $n-1$ different cuts in time that is sub-linear in n, which seems harder; alternatively it may be possible to try significantly fewer than n cuts in each of our distributional orders and still guarantee an approximation bound.

Our sparse cuts computation can be used to identify the crossing edges, which have been used in prior work ([49]) to heuristically improve network search, routing, and connectivity. It will be useful to rigorously show such results with provable guarantees.

Chapter 6

Conclusion and Further Study

In this thesis, we developed fast algorithms for a variety of fundamental distributed network problems, where we used random walks as a key subroutine. We presented a round and message optimal algorithm which can be used to output several random walk samples in a continuous online fashion. The theoretical analysis and comprehensive experimental evaluation highlights the effectiveness and efficiency of our algorithm. Then we developed a fast decentralized algorithm for performing random walks in dynamic networks. Our algorithm is the first-known algorithm that provably speeds up random walks in dynamic networks. Furthermore, we showed a key application of this random walk algorithm for the fundamental problem of information spreading in dynamic networks. We further extend the work to show how random walks can be used in other applications such as PageRank computation and sparse cuts computation in graphs. We presented random walk-based algorithms for computing PageRank and prove strong bounds on the round complexity. We also developed fast distributed algorithms for computing sparse cuts in distributed networks. Our algorithms are fully decentralized, lightweight and easily implementable, and can serve as building blocks in the design of self-aware and self-organizing networks that can monitor and control their own topology.

This thesis highlights the fact that random walk techniques are very useful

in static networks as well as in dynamic networks. We believe that the research presented in this thesis opens up a lot of interesting directions for further study. We have discussed this at the end of each chapter. We like to emphasize here on some of them.

Recall that in this thesis, we develop random walk algorithms which builds on a generic idea of creating many "short random walks" from each node in parallel and then "stitching" them to obtain a random walk of certain length. One may try to use this general approach to develop random walk-based algorithms on different graph models or may be in different dynamic graph models. Recently, several fundamental graph problems are being explored in various dynamic graph models. One may also try to consider somewhat stronger adversarial model and show some non-trivial solutions. As a specific question, it remains open whether the random walk techniques and subsequent bounds presented for a dynamic network are optimal. Other problems in this dynamic model that are worth studying include maintaining spanning tree, sampling node from any distribution (recall that we sample nodes from uniform distribution) or finding spectral quantities etc.

Another interesting problem would be to study distributed random walks in directed graphs. Distributed random walks in directed graphs are mostly unexplored. Designing a sub-linear time random walk simulation algorithm in directed graphs (in CONGEST model) would be a challenging task. The main difficulty is the congestion. A node may be visited many times by random walks compared to other nodes in the network. We do not have an immediate tool to bound the number of visits to a node, unlike in the case of undirected graphs. This is because, we do not have a suitable form of stationary distribution of random walks in directed networks.

Finally, these algorithmic ideas may be useful building blocks in designing fully dynamic self-aware distributed graph systems. It would be interesting to additionally consider total message complexity costs for these algorithms explicitly, even though they are implicitly encapsulated within the local per-edge

bandwidth constraints of the CONGEST model.

List of Publications

1. Distributed Computation of Sparse Cuts, A. Das Sarma, A. R. Molla, G. Pandurangan. In *arXiv: http://arxiv.org/abs/1310.5407*, 2013.

2. Storage and Search in Dynamic Peer-to-Peer Networks, J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, E. Upfal. In *Proceedings of 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)* 2013, pages 53-62, Montreal, Canada.

3. Fast Distributed PageRank Computation, A. Das Sarma, A. R. Molla, G. Pandurangan, E. Upfal. In *Proceedings of 14th International Conference on Distributed Computing and Networking (ICDCN)* 2013, pages 11-26, Mumbai, India. [Invited to Special Issue of *Theoretical Computer Science*, accepted for publication (in press)].

4. Fast Distributed Computation in Dynamic Networks via Random Walks, A. Das Sarma, A. R. Molla, G. Pandurangan. In *Proceedings of 26th International Symposium on Distributed Computing (DISC)* 2012, pages 136-150, Salvador, Brazil.

5. Near-Optimal Random Walk Sampling in Distributed Networks, A. Das Sarma, A. R. Molla, G. Pandurangan. In *Proceedings of 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM)* 2012, pages 2906-2910, Florida, USA. [Invited to *Journal on Self Computing*, accepted for publication (in press)].

Bibliography

[1] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical Review E*, 64(4):046135, 2001.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 218–223, 1979.

[3] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. *Combinatorics, Probability & Computing*, 20(4):481–502, 2011.

[4] R. Andersen, F. R. K. Chung, and K. J. Lang. Local graph partitioning using pagerank vectors. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 475–486, 2006.

[5] J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, and E. Upfal. Search and storage in dynamic peer-to-peer networks. In *Proc. of 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–62, 2013.

[6] J. Augustine, G. Pandurangan, and P. Robinson. Fast byzantine agreement in dynamic networks. In *Proc. of 32nd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 74–83, 2013.

[7] J. Augustine, G. Pandurangan, P. Robinson, and E. Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proc. of ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 551–569, 2012.

[8] C. Avin, M. Borokhovich, K. Censor-Hillel, and Z. Lotker. Order optimal information spreading using algebraic gossip. In *Proc. of 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 363–372, 2011.

[9] C. Avin, M. Koucký, and Z. Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Proc. of 35th Coll. on Automata, Languages and Programming (ICALP)*, pages 121–132, 2008.

[10] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM J. Number. Anal.*, 45(2):890–904, 2007.

[11] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *J. Parallel Distrib. Comput.*, 63(1):97–104, 2003.

[12] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *Proc. of ACM SIGMOD Conference*, pages 973–984, 2011.

[13] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proc. of 36th International Conference on Very Large Databases (VLDB)*, 4:173–184, 2010.

[14] J. Bar-Ilan and D. Zernik. Random leaders and random spanning trees. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, pages 1–12, London, UK, 1989. Springer-Verlag.

[15] H. Baumann, P. Crescenzi, and P. Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proc. of 28th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 260–269, 2009.

[16] P. Berenbrink, J. Czyzowicz, R. Elsässer, and L. Gasieniec. Efficient information exchange in the random phone-call model. In *Proc. of 37th Coll. on Automata, Languages and Programming (ICALP)*, pages 127–138, 2010.

[17] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[18] T. Bernard, A. Bui, and O. Flauzac. Random distributed self-stabilizing structures maintenance. In *Proc. of 3rd International School and Symposium on Advanced Distributed Systems (ISSADS)*, pages 231–240, 2004.

[19] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Trans. Internet Technol.*, 5(1):92–128, Feb. 2005.

[20] M. Borokhovich, C. Avin, and Z. Lotker. Tight bounds for algebraic gossip on graphs. In *Proc. of IEEE International Symposium on Information Theory (ISIT)*, pages 1758–1762, 2010.

[21] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of 7th International World-Wide Web Conference (WWW 1998)*, pages 107–117, 1998.

[22] A. Z. Broder. Generating random spanning trees. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 442–447, 1989.

[23] M. Bui, T. Bernard, D. Sohier, and A. Bui. Random walks in distributed computing: A survey. In *Proc. of 4th International Workshop on Innovative Internet Community Systems (IICS)*, pages 1–14, 2004.

[24] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *Proc. of 10th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW)*, pages 346–359, 2011.

[25] K. Censor-Hillel and H. Shachnai. Fast information spreading in graphs with large weak conductance. *SIAM J. Comput.*, 41(6):1451–1465, 2012.

[26] F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumour spreading and graph conductance. In *Proc. of ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 1657–1663, 2010.

[27] A. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proc. of 27th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 213–222, 2008.

[28] A. Clementi, R. Silvestri, and L. Trevisan. Information spreading in dynamic graphs. In *Proc. of 31th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 37–46, 2012.

[29] M. Cook. Calculation of pagerank over a peer-to-peer network. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.123.9069*, 2004.

[30] B. F. Cooper. Quickly routing searches without having to move content. In *Proc. of 4th International Workshop on Peer-To-Peer Systems (IPTPS)*, pages 163–172, 2005.

[31] C. Cooper, A. M. Frieze, and T. Radzik. Multiple random walks in random regular graphs. *SIAM J. Discrete Math.*, 23(4):1738–1761, 2009.

[32] D. Coppersmith, P. Tetali, and P. Winkler. Collisions among random walks on a graph. *SIAM J. Discret. Math.*, 6(3):363–374, 1993.

[33] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Sparse cut projections in graph streams. In *Proc. of European Symposium on Algorithms (ESA)*, pages 480–491, 2009.

[34] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13, 2011.

[35] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[36] A. Das Sarma, A. R. Molla, and G. Pandurangan. Fast distributed computation in dynamic networks via random walks. In *Proc. of 26th International Symposium on Distributed Computing (DISC)*, pages 136–150, 2012.

[37] A. Das Sarma, A. R. Molla, and G. Pandurangan. Near-optimal random walk sampling in distributed networks. In *Proc. of 31st IEEE International Conference on Computer Communications (IEEE INFOCOM)*, pages 2906–2910, 2012. Also accepted for publication in Journal on Self Computing, (in press).

[38] A. Das Sarma, A. R. Molla, and G. Pandurangan. Distributed computation of sparse cuts. *CoRR*, abs/1310.5407, 2013.

[39] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal. Fast distributed pagerank computation. In *Proc. of 14th International Conference on Distributed Computing and Networking (ICDCN)*, pages 11–26, 2013.

[40] A. Das Sarma, D. Nanongkai, and G. Pandurangan. Fast distributed random walks. In *Proc. of 28th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 161–170, 2009.

[41] A. Das Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Efficient distributed random walks with applications. In *Proc. of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 201–210, 2010.

[42] A. Das Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Distributed random walks. *J. ACM*, 60(1):2, 2013.

[43] S. Deb, M. Médard, and C. Choute. Algebraic gossip: a network coding approach to optimal multiple rumor mongering. *Combinatorics, Probability & Computing*, 14:2486–2507, June 2006.

[44] S. Dolev, E. Schiller, and J. L. Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(7):893–905, 2006. Also in PODC 2002.

[45] S. Dolev and N. Tzachar. Spanders: Distributed spanning expanders. *Dept. of Computer Science, Ben-Gurion University, TR-08-02*, 2007.

[46] C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the complexity of information spreading in dynamic networks. In *Proc. of ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 717–736, 2013.

[47] A. Ferreia. Building a reference combinatorial model for manets. *IEEE Network Magazine*, 18(5):24–29, 2004.

[48] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.

[49] C. Gkantsidis, M. Mihail, and A. Saberi. Throughput and congestion in power-law graphs. In *ACM SIGMETRICS*, pages 148–159, 2003.

[50] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, pages 1526–1537, 2005.

[51] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Perform. Eval.*, 63(3):241–263, 2006. Also in IEEE INFOCOM 2004.

[52] N. Goyal, L. Rademacher, and S. Vempala. Expanders via random spanning trees. In *Proc. of ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 576–585, 2009.

[53] V. Grolmusz. A note on the pagerank of undirected graphs. *CoRR*, abs/1205.1960, 2012.

[54] B. Haeupler. Analyzing network coding gossip made easy. In *Proc. of 43nd Symposium on Theory of Computing (STOC)*, pages 293–302, 2011.

[55] B. Haeupler and D. Karger. Faster information dissemination in dynamic networks via network coding. In *Proc. of 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 381–390, 2011.

[56] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[57] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the web. *Computer Networks*, 31(11-16):1291–1303, 1999.

[58] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proc. of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 119–131, 1990.

[59] G. Iván and V. Grolmusz. When the web meets the cell: using personalized pagerank for analyzing protein interaction networks. *Bioinformatics*, 27(3):405–407, 2011.

[60] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal of Computing*, 18(6):1149–1178, 1989.

[61] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 36–43, 2004.

[62] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, 74(1):70–83, 2008.

[63] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012. Also in PODC 2008.

[64] J. M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 163–170, 2000.

[65] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proc. of 42nd Symposium on Theory of Computing (STOC)*, pages 513–522, 2010.

[66] F. Kuhn, R. Oshman, and Y. Moses. Coordinated consensus in dynamic networks. In *Proc. of 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 1–10, 2011.

[67] A. N. Langville and C. D. Meyer. Survey: Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2003.

[68] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, pages 2133 – 2143, 2003.

[69] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing times.* American Mathematical Society, Providence, RI, USA, 2008.

[70] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proc. of ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 395–406, 2003.

[71] L. Lovász and M. Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 346–354, 1990.

[72] L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. *Random Struct. Algorithms*, 4(4):359–412, 1993.

[73] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of International Conference on Supercomputing (ICS)*, pages 84–95, 2002.

[74] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[75] R. Lyons. Asymptotic enumeration of spanning trees. *Combinatorics, Probability & Computing*, 14(4):491–522, 2005.

[76] D. W. Matula and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113–123, 1990.

[77] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[78] R. Morales and I. Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, page 55, 2007.

[79] R. Morales and I. Gupta. Avcol: Availability-aware information aggregation in large distributed systems under uncollaborative behavior. *Computer Networks*, 53(13):2360–2372, 2009.

[80] R. Morales and I. Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(4):446–459, 2009.

[81] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proc. of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 113–122, 2006.

[82] D. Nanongkai, A. Das Sarma, and G. Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *Proc. of 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 257–266, 2011.

[83] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford InfoLab*, 1999.

[84] G. Pandurangan and M. Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook, Second Edition*. CRC Press, 2009.

[85] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. In *Proc. of 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 492–499, 2001.

[86] D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.

[87] N. Perra and S. Fortunato. Spectral centrality measures in complex networks. *Phys. Rev. E*, 78:036107, Sep 2008.

[88] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active query forwarding in sensor networks. *Ad Hoc Networks*, 3(1):91–113, 2005.

[89] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004.

[90] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, UK, 1994.

[91] J. Wang, J. Liu, and C. Wang. Keyword extraction based on pagerank. In *Proc. of The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 857–864, Berlin, Heidelberg, 2007. Springer-Verlag.

[92] D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 296–303, 1996.

[93] M. Zhong and K. Shen. Random walk based node sampling in self-organizing networks. *Operating Systems Review*, 40(3):49–55, 2006.

[94] M. Zhong, K. Shen, and J. I. Seiferas. Non-uniform random membership management in peer-to-peer networks. In *Proc. of IEEE International Conference on Computer Communications (IEEE INFOCOM)*, pages 1151–1161, 2005.