UNIVERSITY OF CALIFORNIA, SAN DIEGO

Local and Distributed Computation for Large Graphs

A dissertation submitted in partial satisfaction of the
requirements for the degree of Doctor of Philosophy

in

Computer Science

by

Olivia Michelle Simpson

Committee in charge:

Professor Fan Chung Graham, Chair
Professor Ery Arias-Castro
Professor Kamalika Chaudhuri
Professor Sanjoy Dasgupta
Professor Julian McAuley

2016

The Dissertation of Olivia Michelle Simpson is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2016

DEDICATION

To everyone who has encouraged me, especially Kyle.

EPIGRAPH

Song to Onions

They improve everything, pork chops to soup,
And not only that but each onion's a group.

Peel back the skin, delve into tissue
And see how an onion has been blessed with issue.

Every layer produces an ovum:
You think you've got three then you find you've got fovum.

Onion on on-
Ion on onion they run,
Each but the smallest one some onion's mother:
An onion comprises a half-dozen other.

In sum then an onion you could say is less
Than the sum of its parts.
But then I like things that more are than profess–
In food and the arts.

Things pungent, not tony.
I'll take Damon Runyon
Over Antonioni–
Who if an *i* wanders becomes Anti-onion.
I'm anti-baloney.

Although a baloney sandwich would
Right now, with onions, be right good.

And so would sliced onions,
        Chewed with cheese,
Or onions chopped and sprinkled
        Over black-eyed peas:

Black-eyed,
  grey-gravied,
    absorbent of essences,
      eaten on New Year's Eve
        peas.

Roy Blount, Jr.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

This dissertation would not have been possible, or nearly as enjoyable, without the support, guidance, and encouragement of my advisor, Fan Chung Graham. You have been a true inspiration, leader, and friend and I am grateful for the chance to have learned from you.

Thank you to my co-authors Francine Blanchet-Sadri, Emily Clader, C. Seshadhri, Andrew McGregor, and Julian McAuley. It was a pleasure working with each of you and I am proud of what we have contributed to the world.

Thank you to my official and unofficial mentors Francine Blanchet-Sandri, Sesh, Tammy Kolda, David Gleich, Christine Klymko, Goeff Sanders, Ileana Streinu, Tom Weston, Farshid Hajir, David Barrington. I have always felt welcome to come to you with any thoughts, questions, fears, or concerns and you have always responded with support and encouragement.

I have a special thanks for the cool networks kids, Kyle Kloster and Christine Klymko. That thread and your friendships have been life enriching, and at times life saving. We'll form that research group and write all those papers someday.

Thank you to my extended GradWIC family who has gotten me through the best and the worst of it. And to my CSE family for making it fun. And to everyone in San Diego who has taken me in and added a little extra sunshine to my life.

My family - my parents, sisters, friends, and my village - has been with me through the fumbles and the triumphs. I would never have accomplished what I have without your tremendous love and support.

Finally, I owe this to my husband, Kyle. Your faith and pride in me have been the driving force behind this dissertation. I could not have done it without you.

Chapters 2 and 3, in part, are reprints of the material authored by Fan Chung and Olivia Simpson as it is recommended by guest editors to appear in *European Journal of Combinatorics*. The dissertation author was a primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in *Internet Mathematics*. Fan Chung and Olivia Simpson, vol 11(4-5), 2015 pp. 449-471. The dissertation author was a primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in the proceedings of *Algorithms and Models for the Web Graph*. Fan Chung and Olivia Simpson, LNCS 9479, 2015 pp. 177-189. The dissertation author was a primary investigator and author of this paper.

Chapter 7 is currently being prepared for submission for publication of the material. Olivia Simpson, Christine Klymko. The dissertation author was the primary investigator and author of this material.

VITA

2009	Bachelor of Science in Mathematics, University of Massachusetts, Amherst
2014	Masters of Science in Computer Science, University of California, San Diego
2016	Doctor of Philosophy in Computer Science, University of California, San Diego

PUBLICATIONS

Olivia Simpson and Julian McAuley, *Predicting risky behavior in social communities*, MLG, 2016.

Fan Chung and Olivia Simpson, *Distributed Algorithms for Finding Local Clusters Using Heat Kernel Pagerank*, Algorithms and Models for the Web Graph, LNCS 9479, 2015, 177–189.

Fan Chung and Olivia Simpson, *Computing Heat Kernel Pagerank and a Local Clustering Algorithm*, to appear in the European Journal of Combinatorics.

Olivia Simpson, C. Seshadhri, and Andrew McGregor, *Catching the head, tail, and everything in between: a streaming algorithm for the degree distribution*, ICDM, 2015.

Fan Chung and Olivia Simpson, *Solving Local Linear Systems with Boundary Conditions Using Heat Kernel Pagerank*, Internet Mathematics, Vol. 11, Issue 4-5, 2015, 449–471.

Fan Chung and Olivia Simpson, *Solving Linear Systems With Boundary Conditions Using Heat Kernel Pagerank*, Algorithms and Models for the Web Graph, 2013, 203–219.

F. Blanchet-Sadri, E. Clader, and O. Simpson, *Border Correlations of Partial Words*, Theory of Computing Systems, Vol. 47, 2010, 179–195.

ABSTRACT OF THE DISSERTATION

Local and Distributed Computation for Large Graphs

by

Olivia Michelle Simpson

Doctor of Philosophy in Computer Science

University of California, San Diego, 2016

Professor Fan Chung Graham, Chair

Graphs are a powerful and expressive means for storing and working with data. As the demand for fast data analysis increases, data is simultaneously becoming intractably large. To address these space constraints, there is a need for graph algorithms which do not require access to the full graph. We consider such algorithms a number of computational settings. The first is local computation which does not require the state of the full graph, but rather uses the output of small, local queries. We then extend methods in this setting to solve problems for distributed computation, where a graph is stored across processors that can

communicate via communication links in a number of rounds, and a dynamic setting in which the graph is changing over time.

A key tool for computation in each of these settings is random walks. Random walks identify vertices which are central in the graph, a critical subroutine for ranking or clustering algorithms. Random walks on a graph are simple to simulate with prescribed transition probabilities using lightweight, local queries, and have the added benefit of being robust to noisy data.

In this dissertation, we give a quantitative analysis of random walks for local computations on a static, centralized graph and introduce a random walk simulation method for computing a vertex ranking known as the heat kernel pagerank of the graph. We then show how to adapt this method to both the distributed and dynamic setting. With an efficient algorithm for simulating random walks in each of these computational settings, we design fast graph algorithms for modern, flexible computational paradigms.

Chapter 1

Introduction

Our world is complex. As curious beings, it is natural to look for patterns in what is around us. A powerful yet simple way of capturing complexity while preserving structure is defining *relationships* between objects. Graphs have proven to be an extremely expressive means for storing data with relational properties and in fact, many data are best expressed as graphs. Consider one of the most prominent graph representations, that of a social network. Modeling social relationships with a network offers a lens into the qualities of a society by characterizing properties of the graph. Similarly, biological systems, road networks, and communication networks are also well abstracted as graphs. While graphs are a natural way to compute over network data, such data tends to be heterogeneous, massive, and is often changing over time. As the demand for fast data analysis grows, there is a need for graph algorithms that address these constraints.

Graphs are inherently hierarchical, and studying graphs gives a global understanding of the network at large (a biological system, human society, web traffic), while there is also opportunity to gain more intimate knowledge of local areas. Focusing graph analysis on local regions is a meaningful and computationally efficient way to gain knowledge of underlying structure. It is especially important when there is limited information available or there is greater interest in a partial

solution rather than a solution over the full graph. Local algorithms are a class of algorithms which compute properties of small regions of the data, and the best performing local algorithms are those which focus computation on the particular area of interest. In essence, the local algorithms we discuss in this dissertation avoid computing full solutions for partial problems and thereby save unnecessary, expensive computational effort. In a large network, possibly of hundreds of millions of vertices, the algorithms we are dealing with and the solutions we are seeking are usually in terms of the size of the requested output and are independent of the full size of the network.

In this dissertation, we present local algorithms for massive and complex networks. We begin with the problems of detecting local clusters in large graphs, and solving local linear systems. Local cluster detection in a fundamental problem with a huge range of applications where identifying vertices which are "close" in some respect is of interest. Laplacian linear systems are useful for identifying linear relationships between properties of vertices and subset boundaries. For both of these problems we consider graphs which are fixed in time (static) and space (centralized).

In the latter part of this dissertation we extend many of these techniques to graphs in more modern computational settings. Namely, we consider computing over massive and complex networks which are distributed across processors which may communicate over specified links, and networks which are evolving over time. We present algorithms for computing local clusters in a distributed graph, as well as an algorithm for detecting big local changes in an evolving graph.

A key tool for computation in each of these algorithms, and indeed in each of these computational frameworks, is random walks. Random walks identify vertices which are central in a graph with respect to some initialization. They are the

critical ingredient for ensuring that local computation only references local areas of the graph and keep algorithmic running times in terms of the size of the local area of interest. In the results of this dissertation, we take advantage of the simplicity of simulating random walks with lightweight and local queries which do not require the full state of the graph. However, there is tremendous potential for random walks in the data mining space beyond the scope of this work. As random walks do not require the full state of the graph, they are agnostic to network topology and can thereby organically discover structural properties without the benefit of apriori assumptions. Furthermore, they are robust to missing and noisy data. Random walks are effective at capturing large changes among important vertices, while small changes will not affect results much.

In this dissertation, we give a quantitative analysis of random walks for approximating a vertex ranking known as heat kernel pagerank. The random walk-based heat kernel pagerank algorithm is a paramount subroutine for local computation on centralized, distributed, and evolving graphs, and will be a major player for the duration of this work. As such, we begin by introducing random walks on graphs in Chapter 1 and our algorithm for computing heat kernel pagerank in Chapter 2.

1.1 Results in this Dissertation

We present an algorithm for computing the heat kernel pagerank in Chapter 2 which we call `ApproxHKPRseed`. This algorithm is premised around sampling a number of random walks and requires only $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$ random walk steps on a graph of size n to compute significant heat kernel pagerank values within an additive and multiplicative error of ϵ. In Chapter 3 we present an algorithm for finding a local cluster in a graph, `ClusterHKPR`, which calls `ApproxHKPRseed` for

approximating heat kernel pagerank as a subroutine. We show that the ranking induced by heat kernel pagerank will detect clusters with better Cheeger ratio and with better work/volume ratio as compared to algorithms which use alternate rankings. Additionally, we demonstrate that the ranking induced by our random walk approximation algorithm `ApproxHKPRseed` is not too different from the ranking induced by an exact heat kernel pagerank.

We present a random-walk based algorithm for solving local Laplacian linear systems in Chapter 4. This algorithm avoids any matrix computations and returns an approximation of the local solution in sublinear time. We extend the results of this chapter to present an algorithm for computing consensus among a network of agents in Chapter 5 again modeling the network protocol in terms of heat kernel pagerank and computing a solution with random walks.

In Chapter 6 we present algorithms for computing local clusters in graphs that are distributed across processors of a computer network which are allowed to communicate small-sized messages in rounds. Specifically, we present an algorithm which computes a local cluster in $O\left(\frac{\log(\epsilon^{-1})\log n}{\log\log(\epsilon^{-1})} + \frac{1}{\epsilon}\log n\right)$ rounds of communication using heat kernel pagerank where the previously best performing algorithm required $O\left(\frac{1}{\alpha}\log^2 n + n\log n\right)$ rounds.

Finally, in Chapter 7 we present a sampling algorithm which detects large changes on an evolving graph. Specifically, if the rank of a vertex changes by a magnitude of $\delta > 0$ over a span of time τ, given a full history of size T we show that sampling $O(\frac{T}{\tau\delta}\log n)$ random walks is enough to capture this change with high probability.

1.2 Graphs as Operators and Data Structures

At its core, a graph is a structure of pairs of points. This mathematical model has been a useful abstraction for storing data with some well-defined relationship between data points and has been enthusiastically adapted as a fundamental data structure.

Definition 1. A graph, $G = (V, E)$ is a data structure consisting of a set of *vertices*, V, corresponding to data points and a set of *edges* between pairs of vertices, $E \subseteq V \times V$.

When the edges are always between distinct vertices, the graph is said to contain no self-loops. When the edges have direction, the graph is said to be *directed*. Otherwise the graph is *undirected* and the edge notation (v_i, v_j) does not imply an ordering (that is (v_i, v_j) is equivalent to (v_j, v_i)). We denote the size of the graph by $|V| = n$ and $|E| = m$.

For the purpose of this dissertation we will consider undirected graphs which contain no self-loops nor isolated vertices unless otherwise specified.

We may discuss the "size" of a vertex by the number of neighbors it has. That is, if $v_i \in V$ is a vertex in the graph, its neighbors are $\mathcal{N}_i = \{v_j \in V \mid (v_i, v_j) \in E\}$ and the *degree* of vertex v_i is $d_i = |\mathcal{N}_i|$. For an arbitrary vertex v we will sometimes use d_v to denote the degree. The *volume* of a set of vertices $S \subseteq V$ is the total degree of its vertices, $\mathrm{vol}(S) = \sum_{v_i \in S} d_i$. Degree sequences of graphs have important implications in diffusive and connective properties of graphs, which motivate the following definitions concerning graphs induced by subsets of vertices.

For a subset of vertices, $S \subset V$, we refer to the edges whose removal separate S from the rest of the graph S^C as the *edge boundary*, $\partial(S)$ of S, $\partial(S) = \{(v_i, v_j) \in E \mid v_i \in S, v_j \in S^c\}$. Similarly, the *vertex boundary*, $\delta(S)$, is the set of vertices not

in S which border S, $\delta(S) = \{v_i \in S^C \mid v_i \in \mathcal{N}(v_j) \text{ for some } v_j \in S\}$.

1.2.1 Linear Algebra on Graphs

How does one reference a graph? A common representation is a matrix, where rows and columns correspond to vertex indices. For example, the *adjacency matrix*, $A \in \{0,1\}^{n \times n}$, indicates edges by $A_{i,j} = 1$ if and only if $(v_i, v_j) \in E$.

Before going deeper into graph introduction, we will take a moment to fix some notation. When considering a real vector x defined over the vertices of G, we say $x \in \mathbb{R}^n$. Since we will consider vectors both as functions over V and data arrays, we will use $x(v_i)$ to indicate the i^{th} element of the vector x which synonymously references the output of the function on the vertex v_i, and the value of the vector for vertex v_i. The *support* of x is denoted by $\text{supp}(x) = \{v_i \in V : x(v_i) \neq 0\}$. For a subset of vertices $S \subseteq V$, we say $\sigma = |S|$ is the size of S and use $x \in \mathbb{R}^\sigma$ to denote vectors defined over S. When considering a real matrix M defined over V, we say $M \in \mathbb{R}^{n \times n}$. We reference an element (i, j) of a matrix M by M_{ij}. We use M_S to denote the submatrix of M with rows and columns indexed by vertices in S. Namely, $M_S \in \mathbb{R}^{\sigma \times \sigma}$. Similarly, for a vector $x \in \mathbb{R}^n$, we use x_S to mean the subvector of x with entries indexed by vertices in S. We also define $x(S) = \sum_{v_i \in S} x(v_i)$. Finally, we use x^T and M^T to denote vector and matrix transposes, respectively.

Let D be a diagonal matrix whose entries $D_{i,i} = d_i$ are degrees of vertices. Then a matrix with important spectral properties is the *combinatorial Laplacian*, $L = D - A$. The *normalized Laplacian* is the matrix $\mathcal{L} = D^{-1/2} L D^{-1/2}$. It is an operator on the space of functions $f : V \to \mathbb{R}^n$ defined by

$$\mathcal{L}f(v_i) = \frac{1}{\sqrt{d_i}} \sum_{v_j \in \mathcal{N}(v_i)} \left(\frac{f(v_i)}{\sqrt{d_i}} - \frac{f(v_j)}{\sqrt{d_j}} \right).$$

Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of $\mathcal{L}$ and $\mathbb{P}_i$ be the projection to the ith orthonormal eigenvector of $\mathcal{L}$. All of the eigenvalues of $\mathcal{L}$ are nonnegative, and $\lambda_1 = 0$. Then, when restricting to the space orthogonal to $\mathbb{P}_1$ corresponding to λ_1, we have that

$$\mathcal{L} = \sum_{i=2}^{n} \lambda_i \phi_i.$$

When considering a subset S of vertices that induces a connected subgraph, we have

$$\mathcal{L}_S = \sum_{i=1}^{\sigma} \lambda_i \mathbb{P}_i, \tag{1.1}$$

where $\mathcal{L}_S$ is the Laplacian matrix restricted to the rows and columns corresponding to vertices of S, and these λ_i are the eigenvalues of $\mathcal{L}_S$, and $\mathbb{P}_i$ is the projection to the ith orthonormal eigenvector of $\mathcal{L}_S$. Here, we note that each of the λ_i satisfy $0 < \lambda_i \leq 2$.

The normalized Laplacian is the subject of [21]. Here we give a few important facts.

Lemma 1 ([21])**.** *For a graph G on n vertices,*

1. *$\sum_i \lambda_i \leq n$, where equality holds if and only if G has no isolated vertices.*
2. *If G is connected, $\lambda_2 > 0$. If $\lambda_i = 0$ and $\lambda_{i+1} \neq 0$, then G has exactly i connected components.*
3. *For all $i \leq n$, $\lambda_i \leq 2$.*
4. *The set of eigenvalues of a graph is the union of the set of eigevalues its connected components.*

1.2.2 Random Walks

A random walk is a Markov chain on the vertices of the graph where each successive state is a uniform random neighbor of the current state. This models the process of a "random walker" which moves about the graph by randomly choosing edges to traverse.

For some given initial distribution on the vertices, $s \in \mathbb{R}^n$, the distribution after k random walk steps is $s^T P^k$ where P is the transition probability matrix $P = D^{-1}A$. If the graph is connected and non-bipartite, this amounts to the Markov chain being irreducible and aperiodic, and thus there exists a stationary distribution of this process.

In this dissertation we will be considering slight modifications of the standard random walk mentioned above, examining the distributions known as PageRank and heat kernel pagerank. Each of these distributions will depend on the initial distribution. A common initial distribution we will be discussing is that with all probability on a single entry corresponding to one vertex. For this we will use the notation of an indicator vector χ_v:

$$\chi_v(v_i) = \begin{cases} 1 & \text{if } v = v_i \\ 0 & \text{otherwise.} \end{cases}$$

Then if we discuss initial distributions in this way we might set $s := \chi_v$.

Random walks can be used to model different processes on graphs and to discover local graph structure. For instance, we can imagine random walkers simulating the diffusion of heat on a graph. That is, if we choose an initial distribution as χ_v for some fixed vertex v and allow random walkers to move about the graph from this initial distribution, the resulting distribution will model how

heat has dispersed in the graph. We can leverage this idea to induce a ranking on the vertices. That is, the amount of heat that has accumulated at a particular vertex can be used as a measure of how significant this vertex is with respect to the initial distribution. This idea will prove to be useful in a number of the ranking algorithms we discuss.

Chapter 2

Local Computation with Heat Kernel Pagerank

In the quest for unearthing local structure in large graphs, heat kernel pagerank is a powerful tool for designing precise, efficient algorithms with great control over locality.

Heat kernel pagerank was first introduced in [22] as a variant of personalized PageRank [18] and there are many parallels between the two notions. For instance, PageRank can be viewed as a geometric sum of random walks, and the heat kernel pagerank is an exponential sum of random walks. An alternative interpretation of the heat kernel pagerank is related to the heat kernel of a graph as the fundamental solution to the heat equation. As such, it has connections with diffusion and mixing properties of graphs and has been incorporated into a number of graph algorithmic primitives.

In this chapter we define heat kernel pagerank and offer a few meaningful interpretations. In particular, one important consequence of heat kernel pagerank is a mechanism for controlling diffusion on a graph. As such, heat kernel pagerank is a key device for limiting local computation to local areas of interest.

2.1 Previous Work on Computing Heat Kernel and the Approximation of Matrix Exponentials

There is a relatively recent history of efficient heat kernel computation and applications to graph cuts. Orecchia et al. use a variant of heat kernel random walks in their randomized algorithm for computing a cut in a graph with prescribed balance constraints [77]. A key subroutine in the algorithm is a procedure for computing $e^{-M}v$ for a positive semidefinite matrix M and a unit vector v in time $\tilde{O}(m)$. They show how this can be done with a small number of computations of the form $M^{-1}v$ and applying the Spielman-Teng linear solver [88]. Their main result is a randomized algorithm that outputs a balanced cut in time $O(m \operatorname{polylog} n)$. In a follow up paper, Sachdeva and Vishnoi [84] reduce inversion of positive semidefinite matrices to matrix exponentiation, thus proving that matrix exponentiation and matrix inversion are equivalent to polylog factors. In particular, the nearly-linear running time of the balanced separator algorithm depends upon the nearly-linear time Spielman-Teng solver.

Another method for approximating matrix exponentials is given by Kloster and Gleich in [49]. They use a Gauss-Southwell iteration to approximate the Taylor series expansion of the column vector $e^P e_c$ for transition probability matrix P and e_c a standard basis vector. The algorithm runs in sublinear time assuming the maximum degree of the network is $O(\log \log n)$.

A number of Monte Carlo algorithms for PageRank-like computations are given in [9, 17, 36]. An important result of each of these works is that as the number of random walk samples tends to infinity, Monte Carlo-based estimates converge to the true values at an exponential rate. In particular, it is shown in [36]

that the correct ranking can be preserved with a much more conservative number of samples.

2.2 Heat Kernel and Heat Kernel Pagerank

The heat kernel pagerank is related to the *heat kernel* of the graph, $\mathcal{H}_t$, which can be defined as a fundamental solution to the heat equation:

$$\frac{\partial u}{\partial t} = -\mathcal{L}u, \tag{2.1}$$

and given by $\mathcal{H}_t = e^{-t\mathcal{L}}$.

It will be useful to consider a matrix similar to the heat kernel, $H_t = D^{1/2}\mathcal{H}_t D^{-1/2} = e^{-t(I-P)}$. The matrix $\Delta := I - P$ is sometimes known in the literature as the *Laplace operator*. We can view $\mathcal{L}$ and $\mathcal{H}$ as symmetric versions of Δ and H, respectively.

For a given vector $s \in \mathbb{R}^n$ and value $t \in \mathbb{R}^+$, the *heat kernel pagerank* $\rho_{t,s}$ is defined:

$$\begin{aligned} \rho_{t,s} &= s^T H_t \\ &= s^T e^{-t(I-P)}. \end{aligned} \tag{2.2}$$

Note that the heat kernel pagerank will always be understood as a row vector.

We discuss two important properties of heat kernel pagerank. First, we see from (2.1) that the heat kernel pagerank satisfies the equation:

$$\frac{\partial}{\partial t}\rho_{t,s} = -\rho_{t,s}(I - P). \tag{2.3}$$

Second, we can reinterpret the heat kernel pagerank from (2.2) in terms of distribu-

tions of random walks:

$$\rho_{t,s} = \sum_{k=0}^{\infty} e^{-t} \frac{t^k}{k!} s^T P^k. \tag{2.4}$$

Naturally, although the definition holds for any arbitrary vector s, the random walk interpretation only holds for a stochastic vector whose entries all sum to 1. In this case, the vector s is known as the *preference vector* and allows for random walk personalization. For instance, if we assign s to be a stochastic vector with uniform probability over all vertices in the graph, then random walks will reveal global properties.

A more useful choice for the purposes of controlling locality is a vector with probability concentrated on a single "seed" vertex. In this context, the amount of probability which diffuses from the seed can be viewed as a measure of relative importance and the induced ranking by heat kernel pagerank values can be applied in a number of contexts.

We can compare the heat kernel pagerank to the personalized PageRank vector, given by

$$\mathsf{pr}_{\alpha,s} = \alpha \sum_{k=0}^{\infty} (1-\alpha)^k s^T P^k. \tag{2.5}$$

In this definition, α is often called the *jumping* or *reset* constant, meaning that at any step the random walk may jump to a vertex drawn from s with probability α. When preference is given to a single seed vertex, the random walk is "reset" to that vertex. We note that, compared to the personalized PageRank vector, which can be viewed as a geometric sum, we can expect better convergence rates from the heat kernel pagerank, defined as an exponential sum.

2.3 Computing Heat Kernel Pagerank

Here we describe an algorithm for computing heat kernel pagerank. Specifically, we give a probabilistic approximation algorithm for computing a vector that yields a ranking of vertices close to the heat kernel pagerank vector. The approximation algorithm, `ApproxHKPRseed`, works by simulating random walks and computing contributions of these walks for each vertex in the graph. As a matter of simplification, in this chapter we strictly address heat kernel pagerank with a single vertex as a seed – an analogy to Personalized PageRank with total preference given to a single vertex. Note that heat kernel pagerank with a general preference vector is a combination of heat kernel pagerank with a single seed vertex.

We begin our discussion of heat kernel pagerank approximation with an observation. Each term in the infinite series defining heat kernel pagerank in (2.4) is of the form $e^{-t}\frac{t^k}{k!}s^TP^k$ for $k \in [0, \infty)$. The vector s^TP^k is the distribution after k random walk steps with starting distribution s where $s := \chi_v$ for a specified seed vertex v. Then, if we perform k steps of a random walk given by transition probability matrix P from starting distribution s with probability $p_k = e^{-t}\frac{t^k}{k!}$, the heat kernel pagerank can be viewed as the expected distribution of this process.

This suggests a natural way to approximate the heat kernel pagerank. That is, we can obtain a close approximation to the expected distribution with sufficiently many samples. Our algorithm operates as follows. We perform r random walks to approximate the infinite sum, choosing r large enough to bound the error. We also use the fact that very long walks are performed with small probability. As such, we limit the lengths of our random walks by a finite number K. Both r, K depend on a predetermined error bound ϵ.

In our analysis we will use the following definition of an ϵ-approximate

vector.

Definition 2. Let G be a graph on n vertices, and let $s \in \mathbb{R}^n$ be a vector over the vertices of G. Let $\rho_{t,s}$ be the heat kernel pagerank vector according to s and t. Then we say that $\nu \in \mathbb{R}^n$ is an ϵ-*approximate vector* of $\rho_{t,s}$ if

1. for every vertex $v \in V$ in the support of ν,

$$(1-\epsilon)\rho_{t,s}(v) - \epsilon \leq \nu(v) \leq (1+\epsilon)\rho_{t,s}(v),$$

2. for every vertex with $\nu(v) = 0$, it must be that $\rho_{t,s}(v) \leq \epsilon$.

This definition guarantees that for vertices with significant heat kernel pagerank values, we achieve an approximation of the value with an additive and multiplicative error of ϵ. On the other hand, we can be assured that vertices with an approximate value of 0 are not significant in terms of heat kernel pagerank.

In the following algorithm, we approximate $\rho_{t,s}$ by an ϵ-approximate vector which we denote by $\hat{\rho}_{t,s}$. The running time of the algorithm is $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$. The method and complexity of the algorithm, `ApproxHKPRseed`, are similar to the ApproxRow algorithm for personalized PageRank given in [17].

Theorem 1. *Let G be a graph and let v be a vertex of G. Then, the algorithm* ***`ApproxHKPRseed`***(G, t, v, ϵ) *outputs an ϵ-approximate vector $\hat{\rho}_{t,s}$ of the heat kernel pagerank $\rho_{t,s}$ for $0 < \epsilon < 1$ with probability at least $1 - \epsilon$. The running time of* ***`ApproxHKPRseed`*** *is $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$.*

Proof. Consider the random variable which takes on value $s^T P^k$ with probability $p_k = e^{-t}\frac{t^k}{k!}$ for $k \in [0, \infty)$. The expectation of this random variable is exactly $\rho_{t,s}$. Heat kernel pagerank can be understood as a series of distributions of weighted

Algorithm 1. `ApproxHKPRseed`(G, t, v, ϵ)

input: a graph G, $t \in \mathbb{R}^+$, seed vertex $v \in V$, error parameter $0 < \epsilon < 1$.
output: ρ, an ϵ-approximation of $\rho_{t,s}$.

initialize a 0-vector ρ of dimension n, where $n = |V|$
$r \leftarrow \frac{16}{\epsilon^3} \log n$
$K \leftarrow c \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}$ for any choice of $c \geq 1$
for r iterations **do**
    Start
        simulate a P random walk from vertex v where k steps are taken with probability $e^{-t}\frac{t^k}{k!}$ and $k \leq K$
        let v be the last vertex visited in the walk
        $\rho(v) \leftarrow \rho(v) + 1$
    End
end for
return $1/r \cdot \rho$

random walks over the vertices, and the weights are related to the number of steps taken in the walk. The series can be computed by simulating this process, i.e., draw k according to p_k and compute $s^T P^k$ with sufficiently many random walks of length k.

We approximate the infinite sum by limiting the walks to at most K steps. We will take K to be $K = \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}$. These interrupts risk the loss of contribution to the expected value, but can be upper bounded by $\frac{e^{-t}t^K}{K!} \leq \frac{\epsilon}{2}$ provided that $t > K/\log K$. This is within the error bound for an ϵ-approximate heat kernel pagerank. If $t \leq K/\log K$, the expected length of the random walk is

$$\sum_{k=0}^{\infty} \frac{e^{-t}t^k}{k!} \cdot k = t < K/\log K.$$

Thus we can ignore walks of length more than K while maintaining $\rho_{t,s}(v) - \epsilon \leq \hat{\rho}_{t,s}(v) \leq \rho_{t,s}(v)$ for every vertex v.

Next we show how many samples are necessary. For $k \leq K$, our algorithm simulates k random walk steps with probability $e^{-t}\frac{t^k}{k!}$. To be specific, for a fixed

v, let X_k^i be the indicator random variable defined by $X_k^i = 1$ if a random walk beginning from vertex v ends at vertex v_i in k steps. Let X^i be the random variable that considers the random walk process ending at vertex v_i in *at most* k steps. That is, X^i assumes the vector X_k^i with probability $e^{-t}\frac{t^k}{k!}$. Namely, we consider the combined random walk

$$X^i = \sum_{k \leq K} e^{-t}\frac{t^k}{k!}X_k^i.$$

Now, let $\rho(k)_{t,s}$ be the contribution to the heat kernel pagerank vector $\rho_{t,s}$ of walks of length at most k. The expectation of each X^i is $\rho(k)_{t,s}(v_i)$. Then, by Lemma 2,

$$\begin{aligned}\Pr(X^i < (1-\epsilon)\rho(k)_{t,s}(v_i) \cdot r) &< \exp(-\rho(k)_{t,s}(v_i)r\epsilon^2/2) \\ &= \exp(-(8/\epsilon)\rho(k)_{t,s}(v_i)\log n) \\ &< n^{-4}\end{aligned}$$

for every component with $\rho_{t,s}(v_i) > \epsilon$, since then $\rho(k)_{t,s}(v_i) > \epsilon/2$. Similarly,

$$\begin{aligned}\Pr(X^i > (1+\epsilon)\rho(k)_{t,s}(v_i) \cdot r) &< \exp(-\rho(k)_{t,s}(v_i)r\epsilon^2/4) \\ &= \exp(-(4/\epsilon)\rho(k)_{t,s}(v_i)\log n) \\ &< n^{-2}.\end{aligned}$$

We conclude the analysis for the support of $\rho_{t,s}$ by noting that $\hat{\rho}_{t,s}(v_i) = \frac{1}{r}X^i$, and we achieve an ϵ-multiplicative error bound for every vertex v_i with $\rho_{t,s}(v_i) > \epsilon$ with probability at least $1 - O(n^{-2})$.

On the other hand, if $\rho_{t,s}(v_i) \leq \epsilon$, by the third part of Lemma 2, $\Pr(\hat{\rho}_{t,s}(v_i) > 2\epsilon) \leq n^{-8/\epsilon^2}$. We may conclude that, with high probability, $\hat{\rho}_{t,s}(v_i) \leq 2\epsilon$.

For the running time, we use the assumptions that performing a random walk step and drawing from a distribution with bounded support require constant time. These are incorporated in the random walk simulation, which dominates the computation. Therefore, for each of the r rounds, at most K steps of the random walk are simulated, giving a total of $rK = O\left(\frac{16}{\epsilon^3}\log n \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right) = \tilde{O}(1)$ queries. □

The above analysis relies on the usual Chernoff bounds as stated below.

Lemma 2 ([17])**.** *Let X_i be independent Bernoulli random variables with $X = \sum_{i=1}^{r} X_i$. Then,*

1. *for $0 < \epsilon < 1$, $\Pr(X < (1-\epsilon)r\mathbb{E}(X)) < \exp(-\frac{\epsilon^2}{2}r\mathbb{E}(X))$*
2. *for $0 < \epsilon < 1$, $\Pr(X > (1+\epsilon)r\mathbb{E}(X)) < \exp(-\frac{\epsilon^2}{4}r\mathbb{E}(X))$*
3. *for $c \geq 1$, $\Pr(X > (1+c)r\mathbb{E}(X)) < \exp(-\frac{c}{2}r\mathbb{E}(X))$.*

We conclude this chapter with an important note on an implication of our definition of approximation. The algorithm `ApproxHKPRseed` computes estimates of heat kernel pagerank values for vertices with actual heat kernel pagerank values $\geq \epsilon$. There are at most $\frac{1}{\epsilon}$ such vertices, as this is a stochastic vector. The support of an ϵ-approximate heat kernel pagerank vector $\hat{\rho}_{t,s}$ is therefore $N_{\hat{\rho}_{t,s}} := |\text{supp}(\hat{\rho}_{t,s})| \leq \frac{1}{\epsilon}$. A vector with small support offers computational benefits as we will see in subsequent chapters.

Acknowledgements

Chapter 2, in part, is a reprint of the material authored by Fan Chung and Olivia Simpson as it is recommended by guest editors to appear in *European Journal of Combinatorics*. The dissertation author was a primary investigator and author of this paper.

Chapter 3

Detecting Local Clusters

In large networks, many similar elements can be identified to a single, larger entity by the process of clustering. Increasing granularity in massive networks through clustering eases operations on the network. There is a large literature on the problem of identifying clusters in a graph ([19, 45, 59, 60, 70, 85]), and the problem has found many applications. However, in a variation of the graph clustering problem we may only be interested in a single cluster near one element in the graph. For this, local clustering algorithms are of greater use.

As an example, the problem of finding a local cluster arises in protein networks. A protein-protein interaction (PPI) network has undirected edges that represent an interaction between two proteins. Given two PPI networks, the goal of the pairwise alignment problem is to identify an optimal mapping between the networks that best represents a conserved biological function. In [58], a local clustering algorithm is applied from a specified protein to identify a group similar to that protein. Such local alignments are useful for analysis of a particular component of a biological system (rather than at a systems level which will call for a global alignment). Local clustering is also a common tool for identifying communities in a network. A community is loosely defined as a subset of vertices in a graph which are more strongly connected internally than to vertices outside the subset.

Properties of community structure in large, real world networks have been studied in [56], for example, where local clustering algorithms are employed for identifying communities of varying quality.

The goal of a local clustering algorithm is to identify a cluster in a graph near a specified vertex. Using only local structure avoids unnecessary computation over the entire graph. An important consequence of this are running times which are often in terms of the size of the small side of the partition, rather than of the entire graph. The best performing local clustering algorithms use probability diffusion processes over the graph to determine clusters. In this chapter we present an algorithm which identifies a cluster near a specified vertex with simple computations over heat kernel pagerank.

3.1 Previous Work on Local Clustering Algorithms

Local clustering algorithms were introduced in [88], where Spielman and Teng present a nearly-linear time algorithm for finding local clusters with certain balance constraints. Let $\Phi(S)$ denote the cut ratio of a subset S that we will later define as the Cheeger ratio. Then, given a graph and a subset of vertices S such that $\Phi(S) < \phi$ and $\text{vol}(S) \leq \text{vol}(G)/2$, their algorithm finds a set of vertices T such that $\text{vol}(T) \geq \text{vol}(S)/2$ and $\Phi(T) \leq O(\phi^{1/3} \log^{O(1)} n)$ in time $O(m(\log n/\phi)^{O(1)})$. This seminal work incorporates the ideas of Lovász and Simonovitz [61, 62] on isoperimetric properties of random walks, and their algorithm works by simulating truncated random walks on the graph. Spielman and Teng later improve their approximation guarantee to $O(\phi^{1/2} \log^{3/2} n)$ in a revised version of the paper [89].

The algorithm of [88, 89] improves the spectral methods of [33] and a similar expression in [3] which use an eigenvector of the graph Laplacian to partition the

vertices of a graph. However, the local approach of Spielman and Teng allows us to identify focused clusters without investigating the entire graph. For this reason, the running time of this and similar local algorithms are proportional to the size of the small side of the cut, rather than the entire graph.

Andersen et al. [6] give an improved local clustering algorithm using approximate PageRank vectors. For a vertex subset S with Cheeger ratio ϕ and volume ς, they show that a PageRank vector can be used to find a set with Cheeger ratio $O(\phi^{1/2}\log^{1/2}\varsigma)$. Their local clustering algorithm runs in time $O(\phi^{-1}m\log^4 m)$. The analysis of the above process was strengthened in [5] and emphasized that vertices with higher PageRank values will be on the same side of the cut as the starting vertex.

Andersen and Peres [7] later simulate a volume-biased evolving set process to find sparse cuts. Although their approximation guarantee is the same as that of [6], their process yields a better ratio between the computational complexity of the algorithm on a given run and the volume of the output set. They call this value the *work/volume ratio*, and their evolving set algorithm achieves an expected ratio of $O(\phi^{-1/2}\log^{3/2} n)$. This result is improved by Gharan and Trevisan in [38] with an algorithm that finds a set of conductance at most $O(\epsilon^{-1/2}\phi^{1/2})$ and achieves a work/volume ratio of $O(\varsigma^{\epsilon}\phi^{-1/2}\log^2 n)$ for target volume ς and target Cheeger ratio ϕ. The complexity of their algorithm is achieved by running copies of an evolving set process in parallel. A summary of previous results and our contributions are given in Table 3.1.

Table 3.1. Summary of local clustering algorithms

Algorithm	Cheeger ratio of output set	Work/volume ratio
[89]	$O(\phi^{1/2}\log^{3/2} n)$	$O(\phi^{-2}\,\mathrm{polylog}\, n)$
[6]	$O(\phi^{1/2}\log^{1/2} n)$	$O(\phi^{-1}\,\mathrm{polylog}\, n)$
[7]	$O(\phi^{1/2}\log^{1/2} n)$	$O(\phi^{-1/2}\,\mathrm{polylog}\, n)$
[38]	$O(\epsilon^{-1/2}\phi^{1/2})$	$O(\varsigma^{\epsilon}\phi^{-1/2}\,\mathrm{polylog}\, n)$
This work	$O(\phi^{1/2})$	$O(\varsigma^{-1}\epsilon^{-3}\log n\log(\epsilon^{-1})\log\log(\epsilon^{-1}))$

3.2 Using Heat Kernel Pagerank For Finding Local Clusters

The cluster quality of a cut set can be measured by the ratio of the number of edges between the two parts of the cut and the volume of the smaller side of the cut. This is called the *Cheeger ratio* of a set, defined by

$$\Phi(S) = \frac{|\partial(S)|}{\min(\mathrm{vol}(S), \mathrm{vol}(S^C))}.$$

The *Cheeger constant* of a graph is the minimal Cheeger ratio,

$$\Phi(G) = \min_{S\subset G} \Phi(S).$$

For a given subset S of a graph G, the *local Cheeger ratio* is defined

$$\Phi^*(S) = \min_{T\subseteq S} \Phi(T).$$

Our local clustering algorithm is derived from a local version of the usual Cheeger inequalities which relate the Cheeger constant of a graph to an eigenvalue associated to the graph. Namely, for a subset of vertices S with $|S| = \sigma$, define the restricted Laplacian $\mathcal{L}_S = D_S^{-1/2}(D_S - A_S)D_S^{-1/2}$ where D_S and A_S are the restricted matrices of D and A with rows and columns indexed by vertices in S.

Then the eigenvalues $\lambda_S := \lambda_{S,1} \leq \lambda_{S,2} \leq \cdots \leq \lambda_{S,\sigma}$ of $\mathcal{L}_S$ are also known as the *Dirichlet eigenvalues of* S, and are related to $\Phi^*(S)$ by the following local Cheeger inequality [23]:

$$\frac{1}{2}(\Phi^*(S))^2 \leq \lambda_S \leq \Phi^*(S). \tag{3.1}$$

The inequality (3.1) will be used to derive a relationship between a ranking according to heat kernel pagerank and sets with good Cheeger ratios.

The premise of the local clustering algorithm is to find a good cut near a specified vertex by performing a *sweep* over a vector associated to that vertex, which we will specify presently. Let $p \in \mathbb{R}^n$ be a probability distribution vector over the vertices of the graph of support size $N_p := \text{supp}(p)$. Then, consider a probability-per-degree ordering of the vertices where $p(v_1)/d_1 \geq p(v_2)/d_2 \geq \cdots \geq p(v_{N_p})/d_{N_p}$. Let S_i be the set of the first i vertices per the ordering. We call each S_i a *segment*. Then the process of investigating the cuts induced by the segments to find an optimal cut is called performing a sweep over p.

In this section we will show how a sweep over a single heat kernel pagerank finds local clusters. Specifically, we show that for a subset S with $\text{vol}(S) \leq \text{vol}(G)/4$ and $\Phi(S) \leq \phi$, and for a large number of vertices in $v \in S$, performing a sweep over the ϵ-approximate vector $\hat{\rho}_{t,s}$ with v as the seed will find a set with Cheeger ratio at most $O(\sqrt{\phi})$.

The ς-local Cheeger ratio of a sweep over a vector ν is the minimum Cheeger ratio over segments S_i with volume $0 \leq \text{vol}(S_i) \leq 2\varsigma$. Let $\Phi_\varsigma(\nu)$ be the ς-local Cheeger ratio of segments over a sweep of ν that separates sets of volume between 0 and 2ς.

We will use the following bounds for heat kernel pagerank in terms of local Cheeger ratios and sweep cuts to reason that many vertices v can serve as good seeds

for performing a sweep. To be explicit, when we refer to a heat kernel pagerank with preference vector centered around a particular seed v such that $s := \chi_v$, we will use the notation $\rho_{t,v}$.

Lemma 3. *Let G be a graph and S a subset of vertices of volume $\varsigma \leq \mathrm{vol(G)}/4$. Then the set of $v \in S$ satisfying*

$$\frac{1}{2}e^{-t\Phi^*(S)} \leq \rho_{t,v}(S) \leq \sqrt{\varsigma}e^{-t\Phi_\varsigma(\rho_{t,f_S})^2/4}$$

has volume at least $\varsigma/2$.

To proof Lemma 3, we begin with some bounds for heat kernel pagerank in terms of local Cheeger ratios and sweep cuts. For a subset S, define f_S to be the following distribution over the vertices,

$$f_S(v_i) = \begin{cases} d_i/\mathrm{vol}(S) & \text{if } v_i \in S \\ 0 & \text{otherwise.} \end{cases}$$

Then the expected value of $\rho_{t,v}(S)$ over v in S is given by:

$$\begin{aligned} \mathbb{E}(\rho_{t,v}(S)) &= \sum_{v\in S} \frac{d_v}{\mathrm{vol}(S)} \rho_{t,v}(S) \\ &= \sum_{v\in S} \frac{d_v}{\mathrm{vol}(S)} (\chi_v^T H_t)(S) \\ &= f_S^T H_t(S) \\ &= \rho_{t,f_S}(S). \end{aligned} \tag{3.2}$$

We will make use of the following result, given here without proof (see [23]), which bounds the expected value of $\rho_{t,v}(S)$ given by (3.2) in terms of local Cheeger

ratios.

Lemma 4 ([23])**.** *In a graph G, and for a subset S, the following holds:*

$$\frac{1}{2}e^{-t\Phi^*(S)} \leq \frac{1}{2}e^{-t\lambda_S} \leq \rho_{t,f_S}(S).$$

Next, we use an upper bound on the amount of probability remaining in S after sufficient mixing. This is an extension of a theorem given in [23].

Theorem 2. *Let G be a graph and S a subset of vertices with volume $\varsigma \leq \mathrm{vol(G)}/4$. Then,*

$$\rho_{t,f_S}(S) \leq \sqrt{\varsigma}e^{-t\Phi_\varsigma(\rho_{t,f_S})^2/4}.$$

To prove Theorem 2, we define the following for an arbitrary function $f : V \to \mathbb{R}$ and any integer x with $0 \leq x \leq \mathrm{vol}(G)/2$,

$$f(x) = \max_{T\subseteq V\times V, |T|=x} \sum_{(v_i,v_j)\in T} f(v_i, v_j), \quad f(v_i, v_j) = \begin{cases} f(v_i)/d_i, \text{ if } (v_i, v_j) \in E, \\ 0, \text{ otherwise.} \end{cases}$$

The above definition can be extended to all real values of x,

$$f(x) = \max_{T\subseteq V\times V, |T|=x} \sum_{(v_i,v_j)\in T} \alpha_{ij} f(v_i, v_j), \quad \alpha_{ij} \leq 1 \text{ if } (v_i, v_j) \in E, \quad \sum_{(v_i,v_j)\in E} \alpha_{ij} = x.$$

Claim 1. *Let S_i be a segment according to a vector $f \in \mathbb{R}^n$ such that $x = \mathrm{vol(S_i)}$ and $f(v) > 0$ for every $v \in S_i$. Then*

$$f(x) = \sum_{v\in S_i} f(v) = f(S_i).$$

Proof. We are considering the maximum over a subset of vertex pairs T of size

$\text{vol}(S_i)$. Since we are only adding values over vertex pairs which are edges in G, this maximum is achieved when

$$\begin{aligned} f(x) &= \sum_{v \in S_i} \sum_{v_i \in \mathcal{N}(v)} f(v)/d_v \\ &= \sum_{v \in S_i} f(v) \sum_{v_i \in \mathcal{N}(v)} 1/d_v \\ &= f(S_i). \end{aligned}$$

□

Theorem 2. Let Z be the lazy random walk $Z = 1/2(I + P)$. Then,

$$\begin{aligned} f^T Z(S) &= 1/2\Big(f(S) + \sum_{v_j \in \mathcal{N}(v_i), v_i \in S} f(v_i, v_j)\Big) \\ &= 1/2\Big(\sum_{v_i \vee v_j \in S} f(v_i, v_j) + \sum_{v_i \wedge v_j \in S} f(v_i, v_j)\Big) \\ &\leq 1/2(f(\text{vol}(S) + |\partial(S)|) + f(\text{vol}(S) - |\partial(S)|)) \\ &= 1/2\Big(f(\text{vol}(S)(1 + \Phi(S))) + f(\text{vol}(S)(1 - \Phi(S)))\Big). \end{aligned}$$

Let $f_t = \rho_{t,f_S}$, and let x satisfy $0 \leq x \leq 2\varsigma \leq \text{vol}(G)/2$ and represent a volume of some set S_i. Then taking cue from the above inequality, we can associate S to S_i, $\text{vol}(S)$ to $\text{vol}(S_i) = x$ and $\Phi(S)$ to the minimum Cheeger ratio of a set S_i satisfying $\text{vol}(S_i) = x \leq 2\varsigma$, or $\Phi_\varsigma(\rho_{t,f_S})$. Then using Claim 1,

$$f_t Z(x) \leq 1/2(f_t(x(1 + \Phi_\varsigma(\rho_{t,f_S}))) + f_t(x(1 - \Phi_\varsigma(\rho_{t,f_S})))).$$

Now consider the following differential inequality,

$$\begin{aligned}
\frac{\partial}{\partial t} f_t^T(x) &= -\rho_{t,f_S}(I - W)(x) && (3.3)\\
&= -2\rho_{t,f_S}(I - Z)(x)\\
&= -2f_t(x) + 2f_t Z(x)\\
&\leq -2f_t(x) + f_t(x(1 + \Phi_\varsigma(\rho_{t,f_S})))\\
&\quad + f_t(x(1 - \Phi_\varsigma(\rho_{t,f_S})))\\
&\leq 0. && (3.4)
\end{aligned}$$

Line (3.3) follows from (2.3), and line (3.4) follows from the concavity of f.

Consider $g_t(x)$ to be $g_t(x) = \sqrt{x}e^{-t\Phi_\varsigma(\rho_{t,f_S})^2/4}$. Then,

$$\begin{aligned}
&-2g_t(x) + g_t(x(1 + \Phi_\varsigma(\rho_{t,f_S}))) + g_t(x(1 - \Phi_\varsigma(\rho_{t,f_S})))\\
&\quad = -2g_t(x) + \sqrt{1 + \Phi_\varsigma(\rho_{t,f_S})}g_t(x) + \sqrt{1 - \Phi_\varsigma(\rho_{t,f_S})}g_t(x)\\
&\quad = (-2 + \sqrt{1 + \Phi_\varsigma(\rho_{t,f_S})} + \sqrt{1 - \Phi_\varsigma(\rho_{t,f_S})})g_t(x)\\
&\quad \leq \frac{-\Phi_\varsigma(\rho_{t,f_S})^2}{4} g_t(x) && (3.5)\\
&\quad = \frac{\partial}{\partial t} g_t(x),
\end{aligned}$$

where we use the fact that $-2 + \sqrt{1+y} + \sqrt{1-y} \leq -y^2/4$ for $y \in (0, 1]$ in line (3.5). Now, since $f_t(0) = g_t(0)$ and $\frac{\partial}{\partial t} f_t(x)|_{t=0} \leq \frac{\partial}{\partial t} g_t(x)|_{t=0}$,

$$\begin{aligned}
&-2f_t(x) + f_t(x(1 + \Phi_\varsigma(\rho_{t,f_S}))) + f_t(x(1 - \Phi_\varsigma(\rho_{t,f_S})))\\
&< -2g_t(x) + g_t(x(1 + \Phi_\varsigma(\rho_{t,f_S}))) + g_t(x(1 - \Phi_\varsigma(\rho_{t,f_S}))),
\end{aligned}$$

and in particular, $\frac{\partial}{\partial t} f_t(x) \leq \frac{\partial}{\partial t} g_t(x)$. Since $f_0(x) \leq g_0(x)$, we may conclude that

$$f_t(x) \leq g_t(x) = \sqrt{x} e^{-t\Phi_\varsigma(\rho_{t,f_S})^2/4}.$$

□

Using Lemma 4 and Theorem 2, we arrive at the following useful inequalities.

Corollary 1. *Let G be a graph and S a subset with volume $\varsigma \leq \mathrm{vol(G)}/4$. Then,*

$$\frac{1}{2} e^{-t\Phi^*(S)} \leq \rho_{t,f_S}(S) \leq \sqrt{\varsigma} e^{-t\Phi_\varsigma(\rho_{t,f_S})^2/4}.$$

We are now prepared to prove Lemma 3.

Lemma 3. Let F be the set of seeds $F = \{v \in S \ : \ \rho_{t,v}(S) \leq 2\rho_{t,f_S}(S)\}$. Then, by (3.2),

$$F = \{v \in S \ : \ \rho_{t,v}(S) \leq 2\mathbb{E}(\rho_{t,v}(S))\}.$$

Now we consider the set of vertices not included in F,

$$\begin{aligned} \mathbb{E}(\rho_{t,v}(S) \mid v \notin F) &\geq \sum_{v \notin F} \frac{d_v}{\mathrm{vol}(S)} 2\mathbb{E}(\rho_{t,v}(S)) \\ &\geq \frac{\mathrm{vol}(S \setminus F)}{\mathrm{vol}(S)} 2 \sum_{v \notin F} \mathbb{E}(\rho_{t,v}(S)). \end{aligned}$$

Which implies

$$\frac{\mathrm{vol}(S)}{2} > \mathrm{vol}(S \setminus F) \quad \text{or,} \quad \mathrm{vol}(F) > \varsigma/2.$$

□

We can use Lemma 3 to reason that many vertices v satisfy the above inequalities, and thus can serve as good seeds for performing a sweep.

3.3 A Local Graph Clustering Algorithm

It follows from Lemma 3 that the ranking induced by heat kernel pagerank with appropriate seed vertex can be used to find a cut with approximation guarantee $O(\sqrt{\phi})$ by choosing the appropriate t. To obtain a sublinear time local clustering algorithm for massive graphs, we use `ApproxHKPRseed` from Chapter 2 to efficiently compute an ϵ-approximate heat kernel pagerank vector, $\hat{\rho}_{t,v}$, to rank vertices.

The ranking induced by $\hat{\rho}_{t,v}$ is not very different from that of a true vector $\rho_{t,v}$ in the support of $\hat{\rho}_{t,v}$ (for an experimental analysis, see Section 3.4). Namely, using the bounds of Lemma 4, we have

$$\hat{\rho}_{t,v}(S) \geq (1-\epsilon)\rho_{t,v}(S) - \epsilon\sigma.$$

In particular,

$$\frac{1}{2}(1-\epsilon)e^{-t\Phi^*(S)} - \epsilon\sigma \leq \hat{\rho}_{t,v}(S) \leq \sqrt{\varsigma}e^{-t\Phi_\varsigma(\hat{\rho}_{t,v})^2/4} \tag{3.6}$$

for a set of vertices v of volume at least $\varsigma/2$.

Additionally, using $\hat{\rho}_{t,v}$ has the benefit of a small support which eases the work required for a sweep.

Theorem 3. *Let G be a graph and $S \subset G$ a subset with* $\mathrm{vol(S)} = \varsigma \leq \mathrm{vol(G)}/4$, *$|S| = \sigma$, and Cheeger ratio $\Phi(S) \leq \phi$. Let $\hat{\rho}_{t,v}$ be an ϵ-approximate of $\rho_{t,v}$ for some vertex $v \in S$. Then there is a subset $S_t \subset S$ with vol$(S_t) \geq \varsigma/2$ for which a sweep over $\hat{\rho}_{t,v}$ for any vertex $v \in S_t$ with*

1. $t = \phi^{-1}\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$ *and*
2. $\Phi_\varsigma(\hat{\rho}_{t,v})^2 \leq 4/t\log(2)$

finds a set with ς-local Cheeger ratio at most $\sqrt{8\phi}$.

Proof. Let v be a vertex in S_t as described in the theorem statement. Using the inequalities (3.6), we can bound the local Cheeger ratio by a sweep over $\hat{\rho}_{t,v}$:

$$e^{-t\Phi^*(S)} \leq \frac{2}{1-\epsilon}(\sqrt{\varsigma}e^{-t\Phi_\varsigma(\hat{\rho}_{t,v})^2/4} + \epsilon\sigma)$$

which implies

$$e^{-t\Phi^*(S)} \leq e^{-t\Phi_\varsigma(\hat{\rho}_{t,v})^2/4}\left(\frac{2\sqrt{\varsigma}}{1-\epsilon} + \epsilon\sigma e^{t\Phi_\varsigma(\hat{\rho}_{t,v})^2/4}\right),$$

and by the assumption 2, we have

$$\begin{aligned} e^{-t\Phi^*(S)} &\leq e^{-t\Phi_\varsigma(\hat{\rho}_{t,v})^2/4}\left(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma\right) \\ \Phi^*(S) &\geq \frac{\Phi_\varsigma(\hat{\rho}_{t,v})^2}{4} - \frac{\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)}{t}. \end{aligned}$$

Let $x = \log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$. Then,

$$\Phi_\varsigma(\rho_{t,f_S})^2 \leq 4\Phi^*(S) + 4x/t.$$

Since $\Phi^*(S) \leq \Phi(S) \leq \phi$ and $t = \phi^{-1}x$, it follows that $\Phi_\varsigma(\hat{\rho}_{t,v}) \leq \sqrt{8\phi}$. In particular, a sweep over $\hat{\rho}_{t,v}$ finds a cut with Cheeger ratio $O(\sqrt{\phi})$ as long as v is contained in S_t. □

We are now prepared to give our algorithm for finding local clusters with heat kernel pagerank. The algorithm takes as input a starting vertex v, a desired volume ς for the cluster, and a target Cheeger ratio ϕ for the cluster. Then, to find a set achieving a minimum ς-local Cheeger ratio, we perform a sweep over an

approximate heat kernel pagerank vector with the starting vertex as a seed.

Algorithm 2. `ClusterHKPR`$(G, v, \sigma, \varsigma, \phi, \epsilon)$

input: a graph G, a vertex v, target cluster size σ, target cluster volume $\varsigma \leq \text{vol}(G)/4$, target Cheeger ratio ϕ, error parameter ϵ.
output: a set T with $\varsigma/2 \leq \text{vol}(T) \leq 2\varsigma$, $\Phi(T) \leq \sqrt{8\phi}$.

1: $t \leftarrow \phi^{-1} \log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$
2: $\hat{\rho} \leftarrow$ `ApproxHKPRseed`(G, t, v, ϵ)
3: sort the vertices of G in the support of $\hat{\rho}$ according to the ranking $\hat{\rho}(v)/d_v$
4: **for** $j \in [1, n]$ **do**
5: $\quad S_j = \bigcup_{i \leq j} v_i$
6: $\quad$ **if** $\text{vol}(S_j) > 2\varsigma$ **then**
7: $\quad\quad$ output `NO CUT FOUND`, break
8: $\quad$ **else if** $\varsigma/2 \leq \text{vol}(S_j) \leq 2\varsigma$ and $\Phi(S_j) \leq \sqrt{8\phi}$ **then**
9: $\quad\quad$ output S_j
10: $\quad$ **else**
11: $\quad\quad$ output `NO CUT FOUND`
12: $\quad$ **end if**
13: **end for**

Theorem 4. *Let G be a graph which contains a subset S of volume at most* vol(G)/4 *and Cheeger ratio bounded by ϕ. Further, assume that v is contained in the set $S_t \subseteq S$ as defined in Theorem 3. Then* ***`ClusterHKPR`****$(G, v, \sigma, \varsigma, \phi, \epsilon)$ outputs a cutset T with ς-local Cheeger ratio at most $\sqrt{8\phi}$. The running time is the same as that of* ***`ApproxHKPRseed`****.*

Proof. Since it is given that $v \in S_t$ for $t = \phi^{-1} \log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$, and by the assumptions on G and S, Theorem 3 states that a sweep over the approximate heat kernel pagerank $\hat{\rho}$ will find a set with ς-local Cheeger ratio at most $\sqrt{8\phi}$. The checks performed in line 8 of the algorithm discover such a set.

The computational work reduces to the main procedures of computing the heat kernel pagerank vector in line 2 and performing a sweep over the vector in line 4. Performing a sweep involves sorting the support of the vector (line 3) and calculating the conductance of segments. From the guarantees of an ϵ-approximate

heat kernel pagerank vector, any vertex with average probability less than ϵ will be excluded from the support. Then the volume of a vector $\hat{\rho}$ output in line 2 is $O(\epsilon^{-1})$, and performing a sweep over $\hat{\rho}$ can be done in $O(\epsilon^{-1}\log(\epsilon^{-1}))$ time. The algorithm is therefore dominated by the time to compute a heat kernel pagerank vector, and the total running time is $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$. □

3.4 Quantitative Analysis of the Ranking of Vertices with Approximate Heat Kernel Pagerank

The backbone procedure of the local clustering algorithm is the sweep: ranking the vertices of the graph according to their approximate heat kernel pagerank values, and then testing the quality of the cluster obtained by adding vertices one at a time in the order of this ranking. To this end, in this section we compare the rankings of vertices obtained using exact heat kernel pagerank vectors with approximate heat kernel pagerank vectors. Specifically, we consider how accuracy changes as the upper bound of random walk lengths, K, vary.

In the following experiments, we approximate heat kernel pagerank vectors by sampling random walks of length $\min\{k, K\}$, where k is chosen with probability $p_k = e^{-t}\frac{t^k}{k!}$. We test the values computed with different values of K. Since the expected value of a random walk length k chosen with probability $p_k = e^{-t}\frac{t^k}{k!}$ is t, we set K to range from 1 to approximately t for a specified value of t.

In each trial, for a given graph, seed vertex, and value of t, we compute an exact heat kernel pagerank vector $\rho_{t,v}$ and an approximate heat kernel pagerank vector $\hat{\rho}_{t,v}$ using `ApproxHKPRseed` but limiting the length of random walks to K for various K as described above. We then measure how similar the vectors are in

two ways. First, we compare the vector values computed. Second, we compare the rankings obtained with each vector. The following are the measures used:

1. *Comparing vector values.* We measure the error of the approximate vector $\hat{\rho}_{t,v}$ by examining the values computed for each vertex and comparing to an exact vector $\rho_{t,v}$. We use the following measures:

   - **Average L_1 error**: The average absolute error over all vertices of the graph,
   $$\text{average } L_1 \text{ error } := \frac{1}{n}\sum_{i=1}^{n} |\rho_{t,s}(v_i) - \hat{\rho}_{t,s}(v_i)|.$$

   - **ϵ-error**: The accumulated error in excess of an ϵ-approximation (see Definition 2),
   $$\begin{aligned}\epsilon\text{-error } := &\sum_{v_i \in V, \hat{\rho}_{t,s}(v_i) > 0} \max\{|\rho_{t,s}(v_i) - \hat{\rho}_{t,s}(v_i)| - \epsilon\rho_{t,s}(v_i), 0\} \\ &+ \sum_{v_i \in V, \hat{\rho}_{t,s}(v_i) = 0} \max\{\rho_{t,s}(v_i) - \epsilon, 0\}.\end{aligned}$$

2. *Comparing vector rankings.* To measure the similarity of vertex rankings we use the intersection difference (see [13, 34] among others). For a ranked list of vertices A, let A_i be the set of items with the top i rankings. Then we use the following measures:

   - **Intersection difference**: Given two ranked lists of vertices, A and B, each of length n, the intersection difference is
   $$\text{intersection difference} := dist(A, B) = \frac{1}{n}\sum_{i=1}^{n} \frac{|A_i \oplus B_i|}{2i},$$
   where $\oplus$ denotes the symmetric difference $(A_i \setminus B_i) \cup (B_i \setminus A_i)$.

- **Top-k intersection difference**: The intersection difference among the top k elements in each ranking,

$$\text{top-}k \text{ intersection difference} := dist_k(A, B) = \frac{1}{k}\sum_{i=1}^{k}\frac{|A_i \oplus B_i|}{2i}.$$

Intersection difference values lie in the range $[0, 1]$, where a difference of 0 is achieved for identical rankings, and 1 for totally disjoint lists. In these experiments, A is the list of vertices ranked according to an exact heat kernel pagerank vector $\rho_{t,v}$, and B is the list of vertices ranked according to an ϵ-approximate heat kernel pagerank vector $\hat{\rho}_{t,v}$.

In every trial we choose $t = \phi^{-1}\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$ as specified in the local clustering algorithm stated in Section 3.3. This value depends on several parameters, including desired Cheeger ratio, cluster size, and cluster volume. Specifics are provided for each set of trials.

3.4.1 Experiments Illustrating Ranking Accuracy Using Synthetic Graphs

Random graph models

In this series of trials we use three different models of random graph generation provided in the NetworkX [42] Python package, which we describe presently.

The first is the Watts-Strogatz small world model [95], generated with NetworkX. In this model, a ring of n vertices is created and then each vertex is connected to its d nearest neighbors. Then, with probability p, each edge (v_i, v_j) in the original construction is replaced by the edge (v_j, v_k) for a random existing vertex v_k. The model takes parameters n, d, p as input.

The second is the preferential attachment (Barabási-Albert) model [12].

Table 3.2. Random graph models used.

Model	Source	Parameters
small world	Watts-Stragatz[95]	n, number of vertices, d, vertex degree, p, probability of switching an edge.
preferential attachment	Barabási-Albert[12]	n, number of vertices, d, vertex degree
powerlaw cluster	Holme and Kim [43]	n, number of vertices, d, vertex degree, p, probability of forming a triangle

Graphs in this model are created by adding n vertices one at a time, where each new vertex is adjoined to d edges where each edge is chosen with probability proportional to the degree of the neighboring vertex. This is also generated with NetworkX. The model takes parameters n, d as input.

The third NetworkX generator uses the Holme and Kim algorithm for generating graphs with powerlaw degree distribution and approximate average clustering [43]. It is essentially the Barabási-Albert model, but each random edge forms a triangle with another neighbor with probability p. The model takes parameters n, d, p as input.

Table 3.2 lists the random graph models used and their parameters.

Procedure

For every value of K that we test, we generate ten random graphs of each of the three random graph models. For each graph we choose a random seed vertex v with probability proportional to degree, and we choose t as $t = \phi^{-1}\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$ according to the values in Table 3.3. Then for each graph we compare an exact heat kernel pagerank vector $\rho_{t,v}$ and the average of two ϵ-approximate heat kernel pagerank vectors $\hat{\rho}_{t,v}$. The results we present are the average over all trials for each K and each type of graph. We use $d = 5$ and $p = 0.1$ in every trial, and $n = 100$ for

Table 3.3. Parameters used for random graph generation and to compute t for vector computations.

Model	$\|V\|$	d	p	ϵ	Target ϕ	Target σ	Target ς	t
small world	100	5	0.1	0.1	0.05	100	500	84.9
	500	5	0.1	0.1	0.05	100	500	84.9
preferential	100	5	-	0.1	0.05	100	500	84.9
attachment	500	5	-	0.1	0.05	100	500	84.9
powerlaw	100	5	0.1	0.1	0.05	100	500	84.9
cluster	500	5	0.1	0.1	0.05	100	500	84.9

the first set of trials (Figure 3.1) and $n = 500$ for the second (Figure 3.2). These parameters are outlined in Table 3.3.

Discussion

For each graph and value of K, we measure the ϵ-error, the average L_1 error, the intersection difference and the top-10 intersection difference of an approximate heat kernel pagerank vector as compared to an exact heat kernel pagerank vector. Figure 3.1 plots the above measures for graphs over $n = 100$ vertices, while Figure 3.2 plots these measures for graphs over $n = 500$ vertices. In both Figures 3.1 and 3.2, each subplot charts a different notion of error (from top left, clockwise: ϵ-error, average L_1 error, intersection difference and top-10 intersection difference) on the y-axis against K on the x-axis.

In both sets of plots and for every measure of error, we see that in the preferential attachment and powerlaw graphs the error is minimized after limiting random walks to only length $K = 10$, regardless of the size. We observe a shallower decline in ϵ-error, average L_1 error, and intsersection differance for the small world graphs. In particular, we note that the intersection difference drops significantly after 10 random walk steps for all random graphs on both 100 and 500 vertices. For $\epsilon = 0.1$, $K = 4 \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})} \approx 11.04$ is enough to approximate the rankings for

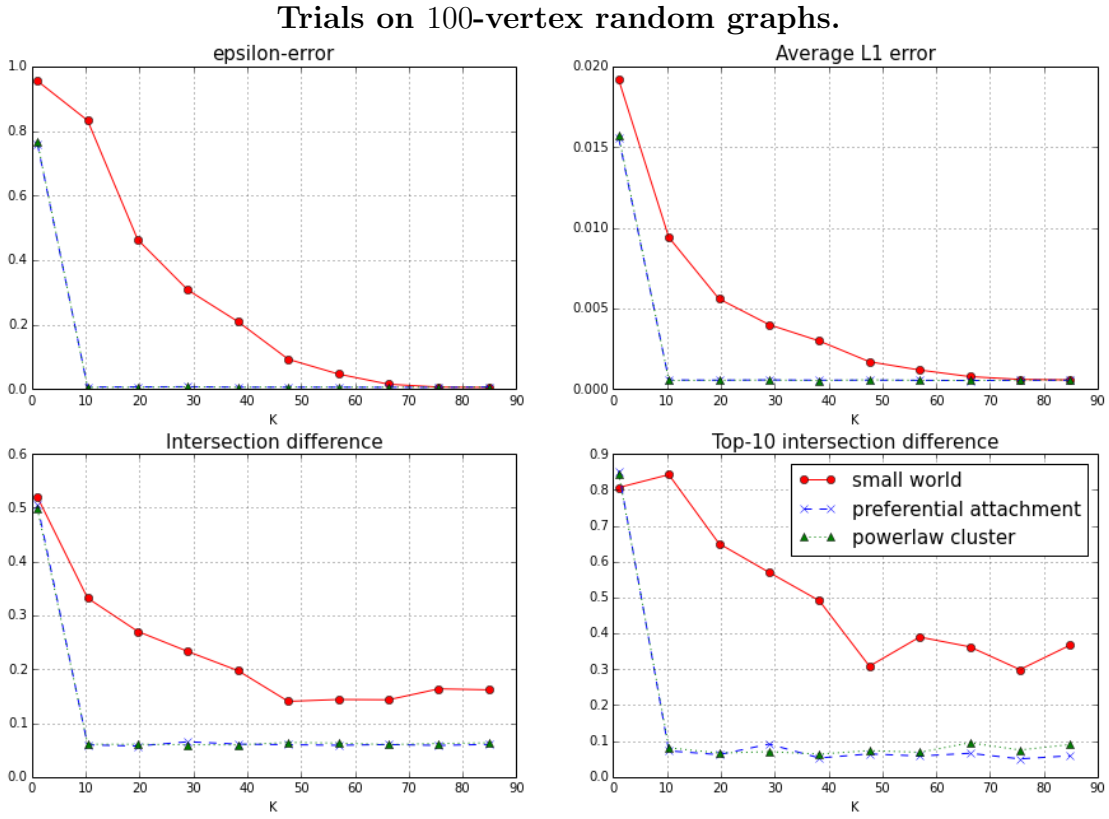

Figure 3.1. Different measures of error for random graphs on 100 vertices when approximating heat kernel pagerank with varying random walk lengths.

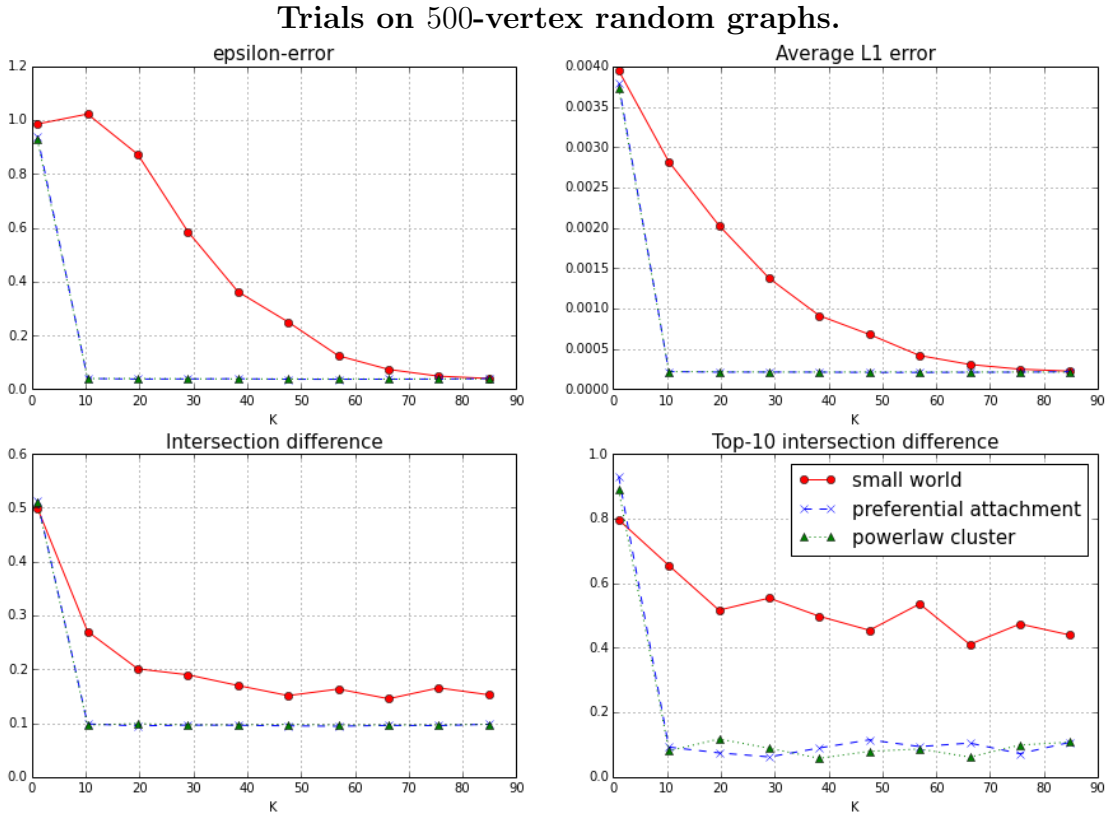

Figure 3.2. Different measures of error for random graphs on 500 when approximating heat kernel pagerank with varying random walk lengths.

the purpose of local clustering.

3.4.2 Experiments Illustrating Ranking Accuracy Using Real Graphs

Network data

For the experiments in this section, and later in Section 3.5, we use the following graphs compiled from real data. The network data is summarized in Table 3.4.

1. **(dolphins)** A dolphin social network consisting of two families [64]. The seed vertex is chosen to be a prominent member of one of the families.

2. **(polbooks)** A network of books about US politics published around the time of the 2004 Presidential election and sold on Amazon [53]. Edges represent frequent copurchases.

3. **(power)** The topology of the US Western States Power Grid [95].

4. **(facebook)** A combined collection of Facebook ego-networks, including the ego vertices themselves [57].

5. **(enron)** An Enron email communication network [48], in which vertices represent email addresses and an edge (v_i, v_j) exists if an address v_i sent at least one email to address v_j.

The network data for graphs 1, 2, and 3 were taken from Mark Newman's network data collection [69]. The network data for graphs 4 and 5 are from the SNAP Large Network Dataset Collection [55].

Table 3.4. Graphs compiled from real data.

Network	Source	$\|V\|$	$\|E\|$	Avg. degree
dolphins	Dolphins social network [64]	62	159	5
polbooks	Political books copurchase network [53]	105	441	8.8
power	Power grid topology [95]	4941	6594	2.7
facebook	Facebook ego-networks [57]	4039	88234	43.7
enron	Enron communication network [48]	36692	183831	10

Table 3.5. Parameters used to compute t for vector computations.

ϵ	Target ϕ	Target σ	Target ς	t
0.1	0.05	100	1000	95.6

Procedure

In this series of experiments, the seed vertex v was chosen to be a known member of a cluster with good Cheeger ratio. As before, t was chosen according to $t = \phi^{-1}\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$ with the values in Table 3.5. For each graph and for each value of K we compare an exact heat kernel pagerank vector $\rho_{t,v}$ with an ϵ-approximate heat kernel pagerank vector $\hat{\rho}_{t,v}$. Specifically, we consider the average L_1 distance and the intersection difference. We again choose K to range from 1 to t.

Discussion

Figure 3.3 plots the average L_1 error on the y-axis against different values of K on the x-axis for each of the dolphins, polbooks, and power graphs. Figure 3.4 plots the intersection difference on the y-axis against K on the x-axis.

First we discuss the average L_1 error. The dolphins and the polbooks graphs exihibit properties of both the small world graphs and the preferential attachment graphs of the previous section (Figures 3.1 and 3.2). Like the preferential attachment

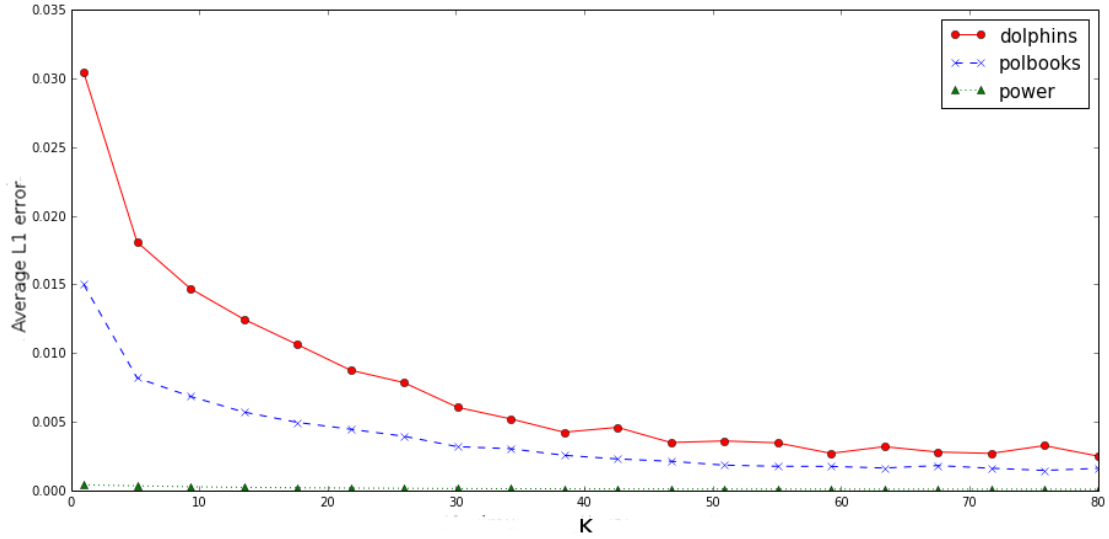

Figure 3.3. Average error in each component for ϵ-approximate heat kernel pagerank vectors when allowing varying random walk lengths.

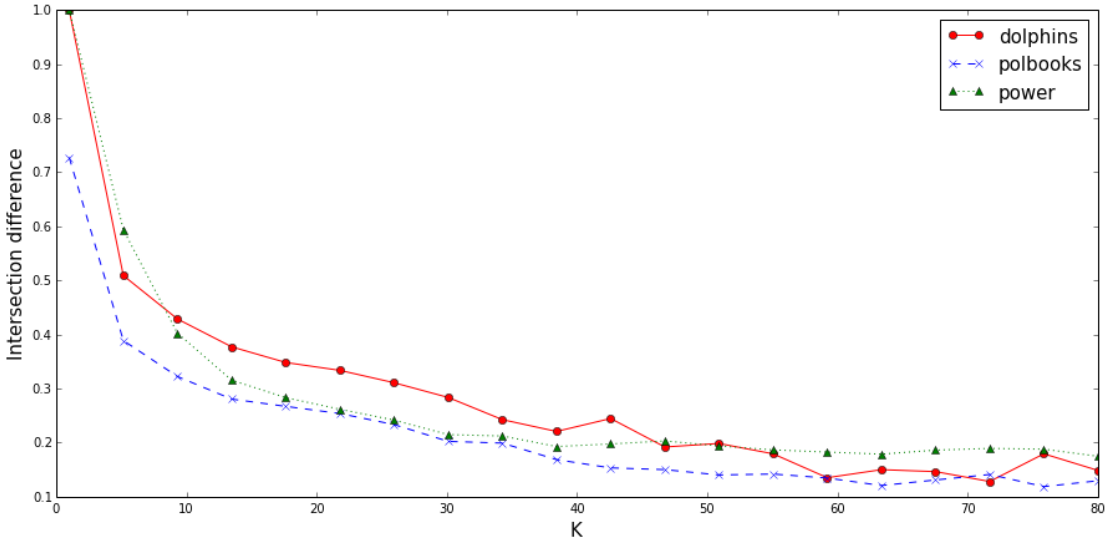

Figure 3.4. Intersersection difference of the ranked lists of vertices computed by exact and ϵ-approximate heat kernel pagerank vectors when allowing varying random walk lengths.

models, there is a significant drop in average L_1 error after $K = 5$, and like the small world model the error continues to drop for larger values of K, approaching a minimum error of ≈ 0.003. The average L_1 error in the power graph, on the other hand, is small for all values of K. We remark that, representing a power grid, the graph has very small average vertex degree, so few random walk steps are enough to approximate the stationary distribution.

As for the intersection difference, we observe a smaller variance in values for the three graphs. Regardless of the size or structure of the graph, the intersection difference drops sharply from $K = 1$ to $K = 5$. For values larger than $K = 10 < 4 \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}$, where $\epsilon = 0.1$, the intersection difference decreases only marginally.

The purpose of these experiments was to evaluate how error and differences of ranking change in heat kernel pagerank approximation when varying K, the upper bound on number of steps taken in random walks. We found that setting an upper bound for random walk lengths to $K = 10 < 4 \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}$ with $\epsilon = 0.1$ according to Theorem 1 yields approximations which satisfy the prescribed error bounds. This value is independent of the size of the graph and t, and depends only on ϵ. Namely, we observed that choosing K this way results in a significant decrease in both average L_1 error and intersection difference as compared to smaller values of K, and only slight decrease in average L_1 error and intersection difference for larger values of K as demonstrated in Figures 3.1, 3.2, 3.3, and 3.4. Further, we tested graphs of various size, random graphs generated from various models, and graphs from real data representing social networks, copurchasing networks, and topological grids. We found this choice of K was optimal for every graph regardless of size or structure. That is, the cutoff for random walk lengths does not depend on the size of the graph.

It is also worth mentioning that the most striking outlier among the subject

graphs is the small world graph, or expander graphs. This is due to the fact that the graph consists of a single cluster, which makes local cluster detection ineffective.

3.5 Performance Analysis of Local Clustering Algorithms

The goal of this section is to analyze the quality of local clusters computed with a sweep over an approximate heat kernel pagerank vector. We consider two objectives for analysis.

The first objective is to validate the statement of Theorem 4. To do this, we show that the Cheeger ratios of local clusters computed with sweeps over approximate heat kernel pagerank vectors are within the approximation guarantees of Theorem 4. We use a slightly modified version of `ClusterHKPR` to compute these clusters. We call this modified algorithm ϵHKPR, and it is described in the list below.

The second objective is to compare clusters computed with sweeps over different vectors. Namely, for a given graph and seed vertex, we compare the local clusters computed using the following sweep algorithms:

1. **(ϵHKPR)** A sweep over an ϵ-approximate heat kernel pagerank vector is performed. The segment S with volume $\text{vol}(S) \leq \text{vol}(G)/2$ of minimal Cheeger ratio is output. This is the `ClusterHKPR` algorithm with the following modification: we allow segments of volume up to $\text{vol}(G)/2$ rather than limiting the search to segments of volume $< 2\varsigma$, twice the target volume.

2. **(HKPR)** A sweep over an exact heat kernel pagerank vector is performed. The segment S with volume $\text{vol}(S) \leq \text{vol}(G)/2$ of minimal Cheeger ratio is output. This algorithm was outlined, but not stated explicitely, in [23].

Table 3.6. Algorithms used for comparing local clusters.

Algorithm	Sweep vector	Algorithm parameters	Sweep parameters
ϵHKPR	$\hat{\rho}_{t,u}$	ϕ, target Cheeger ratio s, target cluster size ς, target cluster volume	$t = \phi^{-1}\log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$ u, seed vertex ϵ, error parameter
HKPR	$\rho_{t,u}$	ϕ, target Cheeger ratio s, target cluster size	$t = 2\phi^{-1}\log s$ u, seed vertex
PR	$\mathsf{pr}_{\alpha,u}$	ϕ, target Cheeger ratio	$\alpha = \phi^2/255\ln(100\sqrt{m})$ u, seed vertex

3. **(PR)** A sweep over a Personalized PageRank vector (2.5) is performed. The segment S with volume $\mathrm{vol}(S) \leq \mathrm{vol}(G)/2$ of minimal Cheeger ratio is output. This is an adaptation of the algorithm `PageRank-Nibble`[6] with the following modifications: (i) rather than performing a sweep over an approximate PageRank vector, perform a sweep over an exact PageRank vector, and (ii) allow segments only as large as $\mathrm{vol}(G)/2$.

We summarize the algorithms and parameters below in Table 3.6.

Each trial will resemble Procedure 3, as stated below.

Procedure 3. Compare Clusters

Let G be a graph and u a seed vertex

Choose parameters ϕ, s, ς, ϵ

Let S_A be a local cluster computed using the algorithm ϵHKPR

Let S_B be a local cluster computed using the algorithm HKPR

Let S_C be a local cluster computed using the algorithm PR

Compare S_A, S_B, S_C.

The following sections describe the experiments in more detail.

Table 3.7. Algorithm parameters used to compare local clusters.

Model	$\|V\|$	d	p	ϵ	Target ϕ	Target σ	Target ς
small world	100	5	0.1	0.1	0.1	20	100
	500	5	0.1	0.1	0.1	100	500
	800	5	0.1	0.1	0.1	100	500
	1000	5	0.1	0.1	0.1	100	500
preferential	100	5	-	0.1	0.1	20	100
attachment	500	5	-	0.1	0.1	100	500
	800	5	-	0.1	0.1	100	500
powerlaw	100	5	0.1	0.1	0.1	20	100
cluster	500	5	0.1	0.1	0.1	100	500
	800	5	0.1	0.1	0.1	100	500

3.5.1 Experiments Illustrating Cheeger Ratio Quality Using Synthetic Graphs

In this section, we use graphs generated with three random graph models: Watts-Strogatz small world, Barabási-Albert preferential attachment, and Holme and Kim's powerlaw cluster as described in Section 3.4.1.

Procedure

We perform twenty-five trials of Procedure 3 and take the averages of Cheeger ratios and cluster volumes computed. Specifically, we fix a model and algorithm parameters. Then, generate a random graph according to the model and parameters. For each random graph, pick a random seed vertex with probability proportional to degree. Then, for each seed vertex compute local clusters S_A, S_B, S_C using the algorithms ϵHKPR, HKPR, and PR, respectively. We then use the average Cheeger ratio and cluster volume of the S_A, S_B, S_C for comparison. In Table 3.7 we summarize the parameters used for each random graph model.

Table 3.8. Cheeger ratios of cluster output by ϵHKPR.

Synthetic graphs

Model	$\lvert V\rvert$	ϕ, Target Cheeger ratio	Cheeger ratio output by ϵHKPR	$\sqrt{8\phi}$
small world	100	0.1	0.173557	0.894427
	500	0.1	0.47316	0.894427
	800	0.1	0.510597	0.894427
	1000	0.1	0.519399	0.894427
preferential attachment	100	0.1	0.523929	0.894427
	500	0.1	0.503542	0.894427
	800	0.1	0.491046	0.894427
powerlaw cluster	100	0.1	0.517521	0.894427
	500	0.1	0.500312	0.894427
	800	0.1	0.494145	0.894427

Discussion

We address the first analytic objective listed in the introduction of this section by discussing the clusters output by ϵHKPR. Namely, we compare the clusters computed with ϵHKPR to the guarantees of Theorem 4. The results for each graph are given in Table 3.8. The first three columns indicate the random graph model and algorithm parameters used for each instance. The last two columns demonstrate how the (average) Cheeger ratio of clusters computed by ϵHKPR compare to the approximation guarantee of Theorem 4. Namely, Theorem 4 states that the cluster output will have Cheeger ratio $\leq \sqrt{8\phi}$ with high probability. In every case the Cheeger ratio is well within the approximation bounds.

The second objective is to compare clusters computed with the three different local clustering algorithms ϵHKPR, HKPR, and PR. Table 3.9 is a collection of cluster statistics for the trials. For each graph instance we list the average Cheeger ratio and cluster volume of local clusters computed using the PR, HKPR, and ϵHKPR algorithms, respectively.

We remark that for each graph there is little variation in Cheeger ratio and

Table 3.9. Cheeger ratios of clusters output by different local clustering algorithms on synthetic data.

Synthetic graphs

Model	$\|V\|$	PR	HKPR	ϵHKPR
small world	100	0.235159 ($\varsigma = 52.52$)	0.087723 ($\varsigma = 171.28$)	0.173557 ($\varsigma = 142$)
	500	0.244261 ($\varsigma = 190.16$)	0.062263 ($\varsigma = 943.68$)	0.47316 ($\varsigma = 206.64$)
	800	0.246564 ($\varsigma = 162.68$)	0.064599 ($\varsigma = 1413.6$)	0.510597 ($\varsigma = 209.6$)
	1000	0.245612 ($\varsigma = 584.56$)	0.064716 ($\varsigma = 1907.4$)	0.519399 ($\varsigma = 225.2$)
preferential attachment	100	0.430071 ($\varsigma = 471.2$)	0.512819 ($\varsigma = 467.16$)	0.523929 ($\varsigma = 468.16$)
	500	0.508305 ($\varsigma = 2461.96$)	0.51018 ($\varsigma = 2459.4$)	0.503542 ($\varsigma = 2463.28$)
	800	0.491046 ($\varsigma = 3964.17$)	0.496369 ($\varsigma = 3971.17$)	0.491046 ($\varsigma = 3951.83$)
powerlaw cluster	100	0.426828 ($\varsigma = 463.4$)	0.505277 ($\varsigma = 465.44$)	0.517521 ($\varsigma = 464.88$)
	500	0.487341 ($\varsigma = 2447.12$)	0.507328 ($\varsigma = 2460.44$)	0.500312 ($\varsigma = 2446.28$)
	800	0.522281 ($\varsigma = 3947$)	0.513365 ($\varsigma = 3966$)	0.494145 ($\varsigma = 3947$)

Table 3.10. Graph and algorithm parameters used to compare local clusters.

Network	$\|V\|$	$\|E\|$	Avg. degree	ϵ	Target ϕ	Target σ	Target ς
dolphins	62	159	5	0.1	0.08	20	100
polbooks	105	441	8.8	0.1	0.05	30	270
power	4941	6594	2.7	0.1	0.05	200	600
facebook	4039	88234	43.7	0.1	0.05	200	2800
enron	36692	183831	10	0.1	0.05	100	1000

volume of clusters computed by the three different algorithms. We also note that there is no obvious trend as graphs get larger. The small world graphs demonstrate the greatest variation in cluster quality. However, as mentioned in Section 3.4, expander graphs, such as small world graphs, consist of one large cluster.

3.5.2 Experiments Illustrating Cheeger Ratio Quality Using Real Graphs

For these trials we use graphs generated from real data summarized in Section 3.4.2.

Procedure

We compare clusters computed by each of the three algorithms as outlined in Procedure 3. In these trials we fix the seed vertex to be a member of a cluster with good Cheeger ratio. Using this seed vertex, we compare the clusters computed using the ϵHKPR, HKPR, PR algorithms.

For each trial we use the parameters listed in Table 3.10. We note that in each case the target cluster volume is computed to be roughly the target cluster size times the average vertex degree, and here we use $\epsilon = 0.1$.

Table 3.11. Cheeger ratios of cluster output `ClusterHKPR`.

Real graphs

Network	ϕ, Target Cheeger ratio	Cheeger ratio output by ϵHKPR	$\sqrt{8\phi}$
dolphins	0.08	0.083333	0.8
polbooks	0.05	0.052133	0.632456
power	0.05	0.346667	0.632456
facebook	0.05	0.056939	0.632456
enron	0.05	0.036602	0.632456

Discussion

Table 3.11 lists ratios output by ϵHKPR compared with the approximation guarantees of Theorem 4. In each case, the Cheeger ratios are well within the approximation bounds of Theorem 4.

The complete numerical data obtained from the set of the trials are given in Table 3.12. For each graph we list the Cheeger ratio, cluster volume, and additionally the cluster size of local clusters computed using each of the algorithms PR, HKPR, and ϵHKPR, respectively.

For each graph, the local cluster computed using ϵHKPR has smaller Cheeger ratio than the local cluster computed using PR. For the power graph, we observe that the cluster of minimal Cheeger ratio was computed using the HKPR algorithm, but it is nearly a third the size of the entire network. The algorithms ϵHKPR and PR, on the other hand, each return smaller clusters. We remark that for real graphs, the clusters computed using sweeps over different vectors have more variation than for random graphs.

To conclude, we include visualizations of clusters computed in the facebook ego-network to illustrate the differences in local cluster detection. Figure 3.5 colors the vertices in a local cluster computed using the ϵHKPR algorithm, as described in Table 3.12. Figure 3.6 colors the vertices in a local cluster compted using the

Table 3.12. Cheeger ratios of cluster output by different local clustering algorithms.

Real graphs

Network	PR	HKPR	ϵHKPR
dolphins	0.226415 ($\varsigma = 106$) ($\sigma = 23$)	0.163636 ($\varsigma = 110$) ($\sigma = 24$)	0.083333 ($\varsigma = 96$) ($\sigma = 20$)
polbooks	0.079518 ($\varsigma = 415$) ($\sigma = 48$)	0.245657 ($\varsigma = 403$) ($\sigma = 49$)	0.052133 ($\varsigma = 422$) ($\sigma = 50$)
power	0.375 ($\varsigma = 16$) ($\sigma = 6$)	0.002764 ($\varsigma = 4342$) ($\sigma = 1564$)	0.346668 ($\varsigma = 300$) ($\sigma = 85$)
facebook	0.418993 ($\varsigma = 88140$) ($\sigma = 3063$)	0.001277 ($\varsigma = 67326$) ($\sigma = 1094$)	0.056939 ($\varsigma = 35266$) ($\sigma = 258$)
enron	0.48797 ($\varsigma = 183612$)	- -	0.036602 ($\varsigma = 3579$)

PR algorithm.

The numerical data of the last two sections validate the effectiveness and efficiency of local cluster detection using sweeps over ϵ-approximate heat kernel pagerank. The experiments of Section 3.4 demonstrate that sampling a number of random walks of at most K steps yield a ranking of vertices within the error bounds of Theorem 1. This ranking in turn is used to compute a local cluster. What is more, this value K does not depend on parameters other than ϵ. Specifically, it does not depend on the size of the graph or the desired cluster volume, size, or Cheeger ratio. Finally, the data of Section 3.5 validate the statements of Theorem 4. That is, perfoming a sweep over an approximate heat kernel pagerank vector detects clusters of Cheeger ratio at most $\sqrt{8\phi}$ for a desired Cheeger ratio ϕ. The total cost of computing this cluster is $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$, sublinear in the size of the graph.

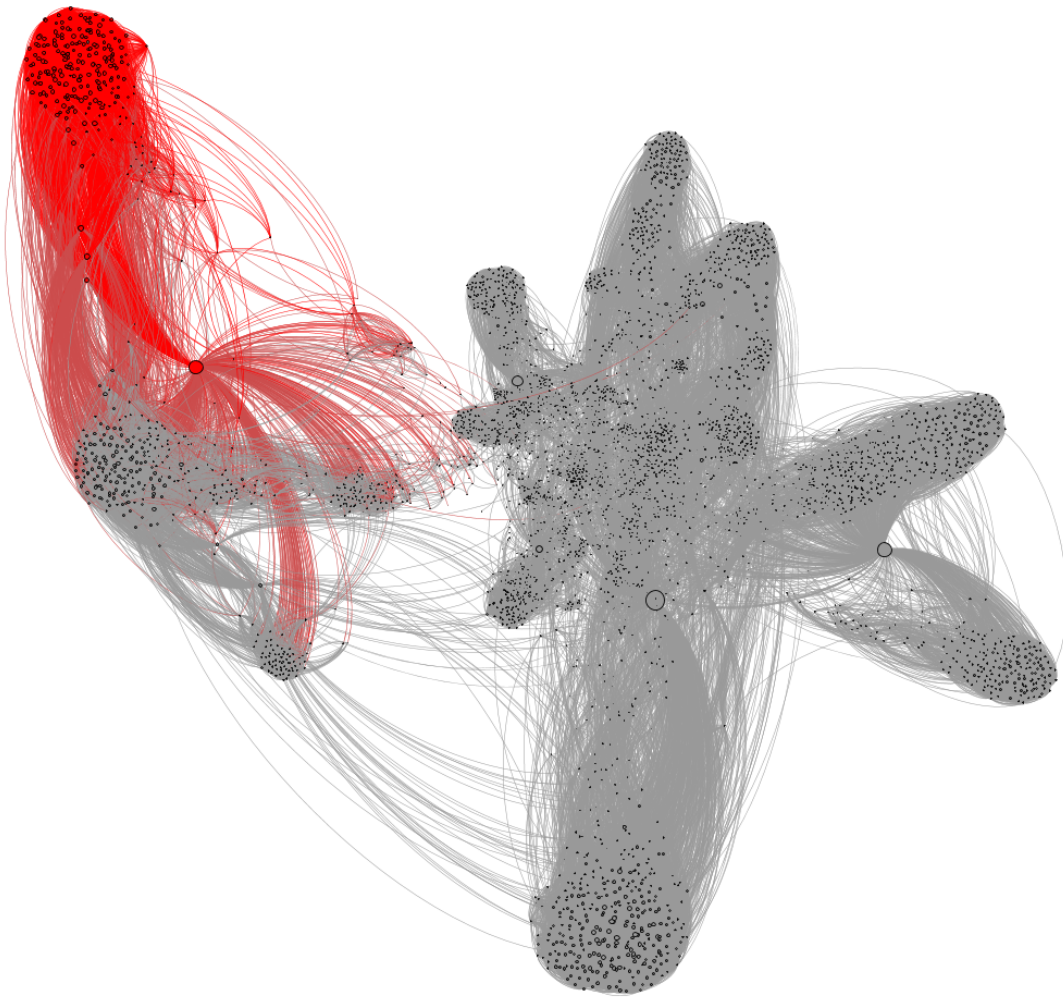

Figure 3.5. Local cluster (colored red) in facebook ego network computed using the ϵHKPR algorithm.

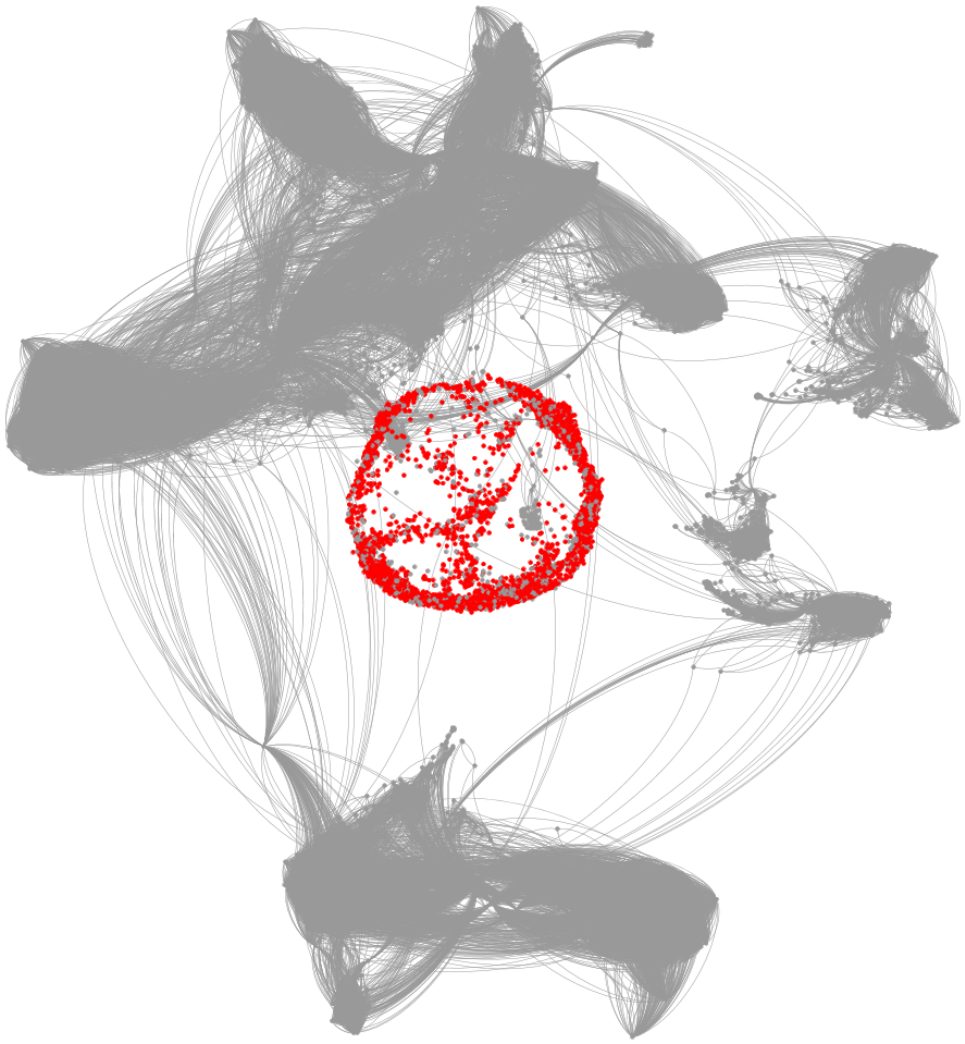

Figure 3.6. Local cluster (colored red) in facebook ego network computed using the PR algorithm.

Acknowledgements

Chapter 3, in part, is a reprint of the material authored by Fan Chung and Olivia Simpson as it is recommended by guest editors to appear in *European Journal of Combinatorics*. The dissertation author was a primary investigator and author of this paper.

Chapter 4

Solving Local Linear Systems on Graphs

There are a number of linear systems which model flow over vertices of a graph with a given boundary condition. A classical example is the case of an electrical network. Flow can be captured by measuring electric current between points in the network, and the amount that is injected and removed from the system. Here, the points at which voltage potential is measured can be represented by vertices in a graph, and edges are associated to the ease with which current passes between two points. The injection and extraction points can be viewed as the boundary of the system, and the relationship of the flow and voltage can be evaluated by solving a system of linear equations over the measurement points.

Another example is a decision-making process among a network of agents. Each agent decides on a value, but may be influenced by the decision of other agents in the network. Over time, the goal is to reach consensus among all the agents, in which each agrees on a common value. Agents are represented by vertices, and each vertex has an associated value. The amount of influence an agent has on a fellow agent is modeled by a weighted edge between the two representative vertices, and the communication dynamics can be modeled by a linear system. In this case,

some special agents which make their own decisions can be viewed as the boundary.

In both these cases, the linear systems are equations formulated in the graph Laplacian. Laplacian systems have been used to concisely characterize qualities such as edge resistance and the influence of communication on edges [86]. There is a substantial body of work on efficient and nearly-linear time solvers for Laplacian linear systems ([14, 26, 37, 46, 50, 51, 52, 54, 81, 84, 88, 93], see also [94]).

The focus of this chapter is a localized version of a Laplacian linear solver. The setup is a graph and a boundary condition given by a vector with specified limited support over the vertices. In the local setting, rather than computing the full solution we compute the solution over a fraction of the graph and de facto ignore the vertices with solution values below the multiplicative/additive error bound. In essence we avoid computing the entire solution by focusing computation on the subset itself. In this way, computation depends on the size of the subset, rather than the size of the full graph. We distinguish the two cases as "global" and "local" linear solvers, respectively. We remark that in the case the solution is not "local," for example, if *all* values are below the error bound, our algorithm will return the zero vector – a valid ϵ-approximate solution.

We show how local Laplacian linear systems with a boundary condition can be solved and efficiently approximated by using a localized version of the heat kernel pagerank called the *Dirichlet heat kernel pagerank*, a diffusion process over an induced subgraph. We will illustrate the connection between the Dirichlet heat kernel pagerank and the Green's function, or the inverse of a submatrix of the Laplacian determined by the subset. Note that in our computation, we do not intend to compute or approximate the matrix form of the inverse of the Laplacian. We intend to compute an approximate local solution which is optimal subject to a particular definition of approximation.

4.1 Previous Work on Solving Laplacian Linear Systems

An early result in scientific computing is the approximation algorithm of the preconditioned Chebyshev method. In [39], Golub and Overton show that for a positive, semi-definite matrix M and a preconditioner B, the preconditioned Chebyshev method finds ϵ-accurate solutions to the system $Mx = b$ in time $O(m\sqrt{\kappa_f(M,B)}S(B)\log(\kappa_f(M)/\epsilon))$. Here, $S(B)$ is the time it takes to solve linear systems in B and

$$\kappa_f(M,B) = \left(\max_{x:Mx\neq 0} \frac{x^T M x}{x^T B x}\right)\left(\max_{x:Mx\neq 0} \frac{x^T B x}{x^T M x}\right).$$

This can be pretty good with a clever choice of preconditioner matrix B.

A large work on finding fast linear solvers for systems of equations in the graph Laplacian was presented by Spielman and Teng in 2004 [88]. Spielman and Teng improve the preconditioned (or inexact) Chebyshev method by exploiting the insight of Vaidya that matrices of subgraphs serve as good preconditioners. In the manuscript [93], Vaidya proves that a maximum spanning tree of a matrix M nm-approximates M and that by adding t^2 edges, one can obtain a sparse graph that $O(nm/t^2)$-approximates M. The result of this is an algorithm for solving symmetric, diagonally dominant (SDD) linear systems with non-positive off-diagonal entries of degree d in time $O((dn)^{1.75}\log(\kappa_f(M)/\epsilon))$, where $\kappa_f(M)$ is the ratio of largest to smallest eigenvalue of M. This is a huge improvement from the previous worst-case $O(nm)$-time bound for the Chebyshev iterative method.

With these tools, Spielman and Teng's major result is an $m\log^{O(1)} n$-time linear solver for SDD matrices where n is the dimension of the matrix and m the number of non-zero entries.

Prior to the contributions of Spielman and Teng, Joshi [44] showed how to recursively apply the results of Vaidya to achieve a $O(n \log(n/\epsilon))$ linear solver. This was improved for planar linear systems by Reif [83] to $O(n^{1+\beta} \log^{O(1)}(\kappa_f(M)/\epsilon))$ for any $\beta > 0$. The above techniques use spanning trees as the preconditioner. Spielman and Teng later improve upon the results of Boman and Hendrickson [15, 16], which apply spanning trees to construct ultra-sparsifiers in time $m^{1.31+o(1)} \log(\kappa_f(M)/\epsilon)$ [87].

Koutis et al. [51] give a nearly optimal linear solver for SDD linear systems; an improvement on the Spielman-Teng linear solver. Their algorithm uses an incremental graph sparsification algorithm as a main tool and outputs an approximate vector $\hat{x}$ satisfying $||\hat{x} - M^+ b||_M < \epsilon ||M^+ b||_M$ in time $O(m \log^2 n \log(1/\epsilon))$. Here M^+ denotes the pseudoinverse of M. In [52], the authors improve this bound to $\tilde{O}(m \log n \log(1/\epsilon))^2$. The faster run time results from an improvement in the incremental sparsifier. As mentioned above, the fastest existing iterative method is $O(m \log^{3/2} n \sqrt{\log \log n} \log(\log n/\epsilon))$-time in the unit-RAM model, due to Lee and Sidford [54]. Their improvement is due to an accelerated randomized coordinate descent method which limits the cost per iteration and also achieves faster convergence.

An early result is the Monte Carlo method for solving linear systems [37], in which the inverse of a matrix is computed by translating the system into a random walk model. Our methods use a similar stochastic process and random sampling procedure. In [84], Sachdeva and Vishnoi show that the inverse of a positive semi-definite matrix can be approximated by taking a weighted sum of matrix exponentials. This method is also closely related to ours. However, rather than taking matrix exponentials explicitly, we are adding small contributions of the matrix exponential multiplied with a specified vector. In this way we avoid explicit

computation of the matrix inverse and spare a final matrix vector multiplication in computing the solution.

4.2 Local Laplacian Linear Systems with a Boundary Condition

The examples of computing current flow in an electrical network and consensus in a network of agents typically require solving linear systems with a boundary condition formulated in the Laplacian L. The problem in the global setting is the solution to $Ly = c$, while the solution y is required to satisfy the boundary condition c in the sense that $y(v) = c(v)$ for every vertex v in the support of c. Because our analysis uses random walks, we use the normalized Laplacian $\mathcal{L}$. We note that the solution x for Laplacian linear equations of the form $\mathcal{L}x = b$ is equivalent to solving $Ly = c$ if we take $y = D^{-1/2}x$ and $c = D^{1/2}b$. Specifically, our local solver computes the solution x restricted to S, denoted x_S, and we do this by way of the discrete Green's function.

Example. To illustrate the local setting, we expand upon the problem of a network of decision-making agents. Consider a communication network of agents in which a certain subset of agents $f \subset V$ are *followers* and an adjacent subset $l \subset V \setminus f$ are *leaders* (see Figure 4.1). Imagine that the decision values of each agent depend on neighbors as usual, but also that the values of the leaders are fixed and will not change. Specifically, let x be a vector of decision values of the agents and suppose every follower v_f continuously adjusts their decision according to the protocol:

$$x(v_f) = x(v_f) - \frac{1}{\sqrt{d_{v_f}}} \sum_{v_i \in \mathcal{N}(v_f)} \frac{x(v_i)}{\sqrt{d_i}},$$

while every leader v_l remains fixed at $b(v_l)$. Then the vector of decision values x

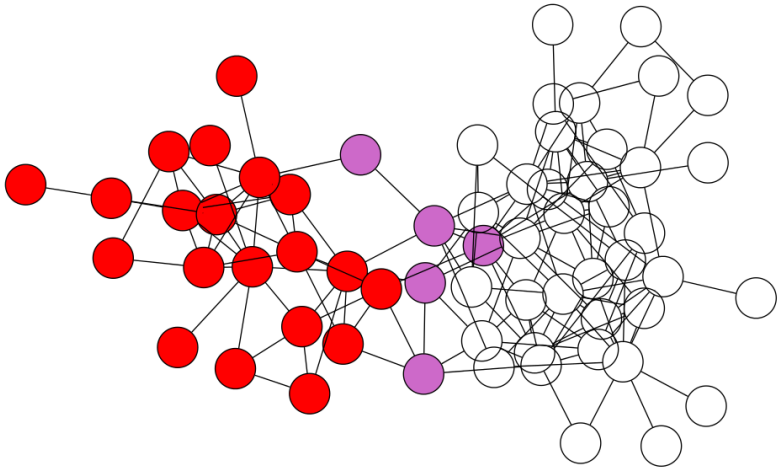

Figure 4.1. A communication network of agents where leaders (in purple) have fixed decisions and followers (in red) compute their decisions. The local solution would be the decisions of the followers.

is the solution to the system $\mathcal{L}x = b$, where x is required to satisfy the boundary condition.

In our example, we are interested in computing the decision values of the followers of the network where the values of the leaders are a fixed boundary condition, but continue to influence the decisions of the subnetwork of followers.

For a general connected, simple graph G and a subset of vertices S, consider the linear system $\mathcal{L}x = b$, where the vector b has non-empty support on the vertex boundary of S. The global problem is finding a solution x that agrees with b, in the sense that $x(v) = b(v)$ for every vertex v in the support of b. In this case we say that x *satisfies the boundary condition* b.

Specifically, for a vector $b \in \mathbb{R}^n$, let S denote a subset of vertices in the complement of supp(b). Then b can be viewed as a function defined on the vertex boundary $\delta(S)$ of S and we say b is a *boundary condition* of S. Here we will consider the case that the induced subgraph on S, denoted G_S, is connected.

Definition 3. Let G be a graph and let b be a vector $b \in \mathbb{R}^n$ over the vertices of G

with non-empty support. Then we say a subset of vertices $S \subset V$ is a *b-boundable subset* if

(i) $S \subseteq V \setminus \text{supp}(b)$,

(ii) $\delta(S) \cap \text{supp}(b) \neq \emptyset$,

(iii) the induced subgraph on S is connected and $\delta(S) \neq \emptyset$.

We remark that in this setup, we *do not* place any condition on b beyond having non-empty support. The entries in b may be positive or negative.

We note that condition (iii) is required in our analysis later, although the general problem of finding a local solution over S can be dealt with by solving the problem on each connected component of the induced subgraph on S individually. It is worth pointing out a number of additional ways our methods can be generalized. First, we focus on unweighted graphs, though extending our results to graphs with edge weights follows easily with a weighted version of the Laplacian. Second, we require the induced subgraph on the subset S be connected. However, if the induced subgraph is not connected the results can be applied to components separately, so our requirement on connectivity can be relaxed. Finally, we restrict our discussion to linear systems in the graph Laplacian. However, by using a linear-time transformation due to [40] for converting a symmetric, diagonally dominant linear system to a Laplacian linear system, our results apply to a larger class of linear systems.

The global solution to the system $\mathcal{L}x = b$ satisfying the boundary condition

b is a vector $x \in \mathbb{R}^n$ with

$$x(v_i) = \begin{cases} \sum_{v_j \in \mathcal{N}(v_i)} \frac{x(v_j)}{\sqrt{d_i d_j}} & \text{if } v_i \in S \\ b(v_i) & \text{if } v_i \notin S \end{cases} \tag{4.1}$$

for a b-boundable subset S. The problem of interest is computing the *local* solution for the *restriction* of x to the subset S, denoted x_S.

Recall that the eigenvalues $\lambda_S := \lambda_{S,1} \leq \lambda_{S,2} \leq \cdots \leq \lambda_{S,\sigma}$ of $\mathcal{L}_S$ are the Dirichlet eigenvalues of S where $\sigma = |S|$. It is easy to check (see [21]) that $0 < \lambda_i \leq 2$ since we assume $\delta(S) \neq \emptyset$. Thus $\mathcal{L}_S^{-1}$ exists and is well defined. Also, assuming the induced subgraph is connected it is also a known fact ([21]) that

$$\lambda_S \geq \frac{1}{diameter(G_S)\text{vol}(S)} \geq \sigma^{-3}. \tag{4.2}$$

However, rarely will graphs come close to achieving this extreme. For example, consider the connected subgraph on S. In this case, the volume will be great, $\sigma(\sigma - 1)$, but the diameter is constant 1. As another extreme, consider the line graph. Here, the diameter is maximized, but the volume is $2\sigma - 2$. In both extreme cases we have that $\lambda_S \geq O(\sigma^{-2})$, and we believe that in most graphs this will be much larger. However, for the remainder of the analysis we use the bounds of (4.2).

Let $A_{S,\delta S}$ be the $\sigma \times |\delta(S)|$ matrix by restricting the columns of A to $\delta(S)$ and rows to S. Requiring S to be a b-boundable subset ensures that the inverse $\mathcal{L}_S^{-1}$ exists [21]. Then the local solution is described exactly in the following theorem.

Theorem 5. *In a graph G, suppose b is a nontrivial vector in $\mathbb{R}^n$ and S is a b-boundable subset. Then the local solution to the linear system $\mathcal{L}x = b$ satisfying*

the boundary condition b satisfies

$$x_S = \mathcal{L}_S^{-1}(D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}). \tag{4.3}$$

Proof. The vector $b_1 := D_S^{-1/2} A_{\delta S} D_{\delta S}^{-1/2} b_{\delta S}$ is defined over the vertices of S by

$$b_1(v_i) = \sum_{v_j \in \mathcal{N}(v_i) \cap \delta(S)} \frac{b(v_j)}{\sqrt{d_i d_j}}. \tag{4.4}$$

Also, the vector $\mathcal{L}_S x_S$ is given by, for $v_i \in S$,

$$\mathcal{L}_S x_S(v_i) = x(v_i) - \sum_{v_j \in \mathcal{N}(v_i) \cap S} \frac{x(v_j)}{\sqrt{d_i d_j}}. \tag{4.5}$$

By (4.1) and (4.3), we have

$$x_S(v_i) = \sum_{v_j \in \mathcal{N}(v_i) \cap S} \frac{x(v_j)}{\sqrt{d_i d_j}} + \sum_{v_j \in \mathcal{N}(v_i) \cap \delta(S)} \frac{b(v_j)}{\sqrt{d_i d_j}},$$

and combining (4.4) and (4.5), we have that $x_S = \mathcal{L}_S^{-1} b_1$. □

4.3 Solving Local Systems with Green's Function

For the remainder of this chapter we are concerned with the local solution x_S. We focus our discussion on the restricted space using the assumptions that the induced subgraph on S is connected and that $\delta(S) \neq \emptyset$. In particular, we consider the *Dirichlet heat kernel*, which is the heat kernel pagerank restricted to S.

The Dirichlet heat kernel is written by $\mathcal{H}_{S,t}$ and is defined as $\mathcal{H}_{S,t} = e^{-t\mathcal{L}_S}$. It is the symmetric version of $H_{S,t}$, where $H_{S,t} = e^{-t(I_S - P_S)} = D_S^{-1/2} \mathcal{H}_{S,t} D_S^{1/2}$.

Recall from (1.1) that the spectral decomposition of $\mathcal{L}_S$ is

$$\mathcal{L}_S = \sum_{i=1}^{\sigma} \lambda_i \mathbb{P}_i,$$

where $\mathbb{P}_i$ are the projections to the i^{th} orthonormal eigenvectors. The Dirichlet heat kernel can be expressed as

$$\mathcal{H}_{S,t} = \sum_{i=1}^{\sigma} e^{-t\lambda_i} \mathbb{P}_i.$$

Let $\mathcal{G}$ denote the inverse of $\mathcal{L}_S$. Namely, $\mathcal{G}\mathcal{L}_S = \mathcal{L}_S\mathcal{G} = I_S$. Then

$$\mathcal{G} = \sum_{i=1}^{\sigma} \frac{1}{\lambda_i} \mathbb{P}_i.$$

Then we see that

$$\frac{1}{2} \leq ||\mathcal{G}|| \leq \frac{1}{\lambda_S},$$

where $||\cdot||$ denotes the spectral norm. We call $\mathcal{G}$ the *Green's function*, and $\mathcal{G}$ can be related to $\mathcal{H}_{S,t}$ as in the following lemma.

Lemma 5. *Let $\mathcal{G}$ be the Green's function of a connected induced subgraph on $S \subset V$ with $\sigma = |S|$. Let $\mathcal{H}_{S,t}$ be the Dirichlet heat kernel with respect to S. Then*

$$\mathcal{G} = \int_0^{\infty} \mathcal{H}_{S,t} \, \mathrm{d}t.$$

Proof. By our definition of the heat kernel,

$$\begin{aligned}\int_0^\infty \mathcal{H}_{S,t}\ \mathrm{d}t &= \int_0^\infty \Big(\sum_{i=1}^{\sigma} e^{-t\lambda_i}\mathbb{P}_i\Big)\mathrm{d}t \\ &= \sum_{i=1}^{\sigma}\Big(\int_0^\infty e^{-t\lambda_i}\,\mathrm{d}t\Big)\mathbb{P}_i \\ &= \sum_{i=1}^{\sigma}\frac{1}{\lambda_i}\mathbb{P}_i \\ &= \mathcal{G}.\end{aligned}$$

□

Equipped with the Green's function, the solution (4.3) can be expressed in terms of the Dirichlet heat kernel. As a corollary to Theorem 5 we have the following.

Corollary 2. *In a graph G, suppose b is a nontrivial vector in $\mathbb{R}^n$ and S is a b-boundable subset. Then the local solution to the linear system $\mathcal{L}x = b$ satisfying the boundary condition b can be written as*

$$x_S = \int_0^\infty \mathcal{H}_{S,t}b_1\ \mathrm{d}t, \tag{4.6}$$

where $b_1 = D_S^{-1/2}A_{S,\delta S}D_{\delta S}^{-1/2}b_{\delta S}$.

The computation of b_1 takes time proportional to the size of the edge boundary.

4.4 A Local Linear Solver Algorithm with Dirichlet Heat Kernel Pagerank

In the previous section, we saw how the local solution x_S to the system satisfying the boundary condition b can be expressed in terms of integrals of Dirichlet heat kernel in (4.6). In this section, we will show how these integrals can be well-approximated by sampling a finite number of values of Dirichlet heat kernel (Theorem 6) and Dirichlet heat kernel pagerank (Corollary 3). All norms $||\cdot||$ in this section are the spectral (L_2) norm.

Theorem 6. *Let G be a graph and $\mathcal{L}$ denote the normalized Laplacian of G. Let b be a nontrivial vector $b \in \mathbb{R}^n$ and S a b-boundable subset, and let $b_1 = D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$. Then the local solution x_S to the linear system $\mathcal{L}x = b$ satisfying the boundary condition b can be computed by sampling $\mathcal{H}_{S,t}b_1$ for $r = \gamma^{-2}\log(\sigma\gamma^{-1})$ values. If $\hat{x}_S$ is the output of this process, the result has error bounded by*

$$||x_S - \hat{x}_S|| = O\big(\gamma(||b_1|| + ||x_S||)\big)$$

with probability at least $1-\gamma$.

We prove Theorem 6 in two steps. First, we show how the integral (4.6) can be expressed as a finite Riemann sum without incurring much loss of accuracy in Lemma 6. Second, we show in Lemma 7 how this finite sum can be well-approximated by its expected value using a concentration inequality.

Lemma 6. *Let x_S be the local solution to the linear system $\mathcal{L}x = b$ satisfying the boundary condition b given in (4.6). Then, for $T = \sigma^3\log(\sigma^3\gamma^{-1})$ and $R = T/\gamma$,*

the error incurred by taking a right Riemann sum is

$$||x_S - \sum_{j=1}^{R} \mathcal{H}_{S,jT/R} \frac{T}{R} b_1|| \leq \gamma(||b_1|| + ||x_S||),$$

where $b_1 = D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$.

Proof. First, we see that:

$$\begin{aligned} ||\mathcal{H}_{S,t}|| &= ||\sum_i e^{-t\lambda_i} \mathbb{P}_i|| \\ &\leq e^{-t\lambda_S} ||\sum_i \mathbb{P}_i|| \\ &= e^{-t\lambda_S} \end{aligned} \tag{4.7}$$

where λ_i are Dirichlet eigenvalues for the induced subgraph S. So the error incurred by taking a definite integral up to $t = T$ to approximate the inverse is the difference

$$\begin{aligned} ||x_S - \int_0^T \mathcal{H}_{S,t} b_1 \, \mathrm{d}t|| &= ||\int_T^\infty \mathcal{H}_{S,t} b_1 \, \mathrm{d}t|| \\ &\leq \int_T^\infty e^{-t\lambda_S} ||b_1|| \, \mathrm{d}t \\ &\leq \frac{1}{\lambda_S} e^{-T\lambda_S} ||b_1||. \end{aligned}$$

Then by the assumption on T the error is bounded by $||x_S - \int_0^T \mathcal{H}_{S,t} b_1 \, \mathrm{d}t|| \leq \gamma ||b_1||$.

Next, we approximate the definite integral in $[0, T]$ by discretizing it. That is, for a given γ, we choose $R = T/\gamma$ and divide the interval $[0, T]$ into R intervals

of size T/R. Then a finite Riemann sum is close to the definite integral:

$$\begin{aligned} || \int_0^T \mathcal{H}_{S,t} b_1 \, \mathrm{d}t - \sum_{j=1}^{R} \mathcal{H}_{S,jT/R} b_1 \frac{T}{R} || &\leq \gamma || \int_0^T \mathcal{H}_{S,t} b_1 \, \mathrm{d}t || \\ &\leq \gamma ||x_S||. \end{aligned}$$

This gives a total error bounded by $\gamma(||b_1|| + ||x_S||)$. □

Lemma 7. *The sum* $\sum_{j=1}^{R} \mathcal{H}_{S,jT/R} b_1 \frac{T}{R}$ *can be approximated by sampling* $\gamma^{-2} \log(\sigma\gamma^{-1})$ *values of* $\mathcal{H}_{S,jT/R} b_1$ *where* j *is drawn from* $[1, R]$*. With probability at least* $1 - \gamma$*, the result has multiplicative error at most* γ*.*

A main tool in our proof of Lemma 7 is the following matrix concentration inequality (see [24], also variations in [2, 28, 41, 76, 92]).

Theorem 7 ([24])**.** *Let* $X_1, X_2, \ldots, X_m$ *be independent random* $n \times n$ *Hermitian matrices. Moreover, assume that* $\|X_i - \mathbb{E}(X_i)\| \leq M$ *for all* i*, and put* $v^2 = \| \sum_i var(X_i) \|$*. Let* $X = \sum_i X_i$*. Then for any* $a > 0$*,*

$$\Pr(\|X - \mathbb{E}(X)\| > a) \leq 2n \exp\left(-\frac{a^2}{2v^2 + 2Ma/3} \right),$$

where $|| \cdot ||$ *denotes the spectral norm.*

Proof of Lemma 7. Suppose without loss of generality that $||b_1|| = 1$. Let Y be a random variable that takes on the vector $\mathcal{H}_{S,jT/R} b_1$ for every $j \in [1, R]$ with probability $1/R$. Then $\mathbb{E}(Y) = \frac{1}{R} \sum_{j=1}^{R} \mathcal{H}_{S,jT/R} b_1$. Let $X = \sum_{i=1}^{r} X_j$ where each X_j is a copy of Y, so that $\mathbb{E}(X) = r\mathbb{E}(Y)$.

Now consider $\mathbb{Y}$ to be the random variable that takes on the projection matrix $\mathcal{H}_{S,jT/R} b_1 (\mathcal{H}_{S,jT/R} b_1)^T$ for every $j \in [1, R]$ with probability $1/R$, and $\mathbb{X}$ is

the sum of r copies of $\mathbb{Y}$. Then we evaluate the expected value and variance of $\mathbb{X}$ as follows:

$$\begin{aligned} ||\mathbb{E}(\mathbb{X})|| &= r||\mathbb{E}(\mathbb{Y})|| \\ ||\mathrm{Var}(\mathbb{X})|| &= r||\mathrm{Var}(\mathbb{Y})|| \leq ||\frac{r}{R}\sum_{j=1}^{R}\mathcal{H}_{S,jT/R}b_1(\mathcal{H}_{S,jT/R}b_1)^T||\mathcal{H}_{S,jT/R}b_1||^2|| \\ &\leq r||\mathbb{E}(\mathbb{Y})||. \end{aligned}$$

We now apply Theorem 7 to $\mathbb{X}$. We have

$$\begin{aligned} \Pr\big(||\mathbb{X}-\mathbb{E}(\mathbb{X})|| \geq \gamma||\mathbb{E}(\mathbb{X})||\big) &\leq 2\sigma\exp\left(-\frac{\gamma^2||\mathbb{E}(\mathbb{X})||^2}{2\mathrm{Var}(\mathbb{X})+\frac{2\gamma||\mathbb{E}(\mathbb{X})||M}{3}}\right) \\ &\leq 2\sigma\exp\left(-\frac{\gamma^2 r^2||\mathbb{E}(\mathbb{Y})||}{r+2\gamma rM/3}\right) \\ &\leq 2\sigma\exp\left(-\frac{\gamma^2 r}{2}\right). \end{aligned}$$

Therefore we have $\Pr\big(||\mathbb{X}-\mathbb{E}(\mathbb{X})|| \geq \gamma||\mathbb{E}(\mathbb{X})||\big) \leq \gamma$ if we choose $r \geq \gamma^{-2}\log(\sigma\gamma^{-1})$. Further, this implies the looser bound:

$$\Pr\big(||X-\mathbb{E}(X)|| \geq \gamma||\mathbb{E}(X)||\big) \leq \gamma.$$

Then $\mathbb{E}(Y) = \frac{1}{r}\mathbb{E}(X)$ is close to $\frac{1}{r}X$ and

$$\begin{aligned} ||\sum_{j=1}^{R}\mathcal{H}_{S,jT/N}b_1\frac{1}{R}-\frac{1}{r}X|| &\leq \gamma||\sum_{j=1}^{R}\mathcal{H}_{S,jT/R}b_1\frac{1}{R}|| \\ ||\sum_{j=1}^{N}\mathcal{H}_{S,jT/R}b_1\frac{T}{R}-\frac{T}{r}X|| &\leq \gamma||\sum_{j=1}^{N}\mathcal{H}_{S,jT/R}b_1\frac{T}{R}|| \end{aligned}$$

with probability at least $1-\gamma$, as claimed. □

Proof of Theorem 6. Let X be the sum of r samples of $\mathcal{H}_{S,jT/R}b_1$ with j drawn

from $[0, R]$, and let $\hat{x}_S = \frac{T}{r}X$. Then combining Lemmas 6 and 7, we have

$$||x_S - \hat{x}_S|| \leq \gamma\big(||b_1|| + ||x_S|| + ||\sum_{j=1}^{R} \mathcal{H}_{S,jT/R} b_1 \frac{T}{R}||\big)$$
$$\leq O\big(\gamma(||b_1|| + ||x_S||)\big).$$

By Lemma 7, this bound holds with probability at least $1 - \gamma$. □

The above analysis allows us to approximate the solution x_S by sampling $\mathcal{H}_{S,t}b_1$ for various t. The following corollary is similar to Theorem 6 except we use the asymmetric version of the Dirichlet heat kernel which we will need later for using random walks. In particular, we use Dirichlet heat kernel pagerank vectors. Dirichlet heat kernel pagerank is also defined in terms of a subset S whose induced subgraph is connected, and a vector $s \in \mathbb{R}^\sigma$ by the following:

$$\rho_{S,t,s} = s^T H_{S,t}. \tag{4.8}$$

Corollary 3. *Let G be a graph and $\mathcal{L}$ denote the normalized Laplacian of G. Let b be a nontrivial vector $b \in \mathbb{R}^n$ and S be a b-boundable subset. Let $b_2 = (D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S})^T D_S^{1/2}$. Then the local solution x_S to the linear system $\mathcal{L}x = b$ satisfying the boundary condition b can be computed by sampling ρ_{S,t,b_2} for $r = \gamma^{-2}\log(\sigma\gamma^{-1})$ values. If $\hat{x}_S$ is the output of this process, the result has error bounded by*

$$||x_S - \hat{x}_S|| = O\big(\gamma(||b_1|| + ||x_S||)\big),$$

where $b_1 = D_S^{-1/2} A_{\delta S} D_{\delta S}^{-1/2} b_{\delta S}$, with probability at least $1 - \gamma$.

Proof. First, we show how x_S can be given in terms of Dirichlet heat kernel

pagerank.

$$\begin{aligned} x_S^T &= \int_0^\infty b_1^T \mathcal{H}_{S,t} \, \mathrm{d}t \\ &= \int_0^\infty b_1^T (D_S^{1/2} H_{S,t} D_S^{-1/2}) \, \mathrm{d}t \\ &= \int_0^\infty b_2 H_{S,t} D_S^{-1/2} \, \mathrm{d}t, \quad \text{where } b_2 = b_1^T D_S^{1/2} \\ &= \int_0^\infty \rho_{S,t,b_2} \, \mathrm{d}t \; D_S^{-1/2}, \end{aligned}$$

and we have an expression similar to (4.6). Then by Lemma 6, x_S^T is close to $\sum_{j=1}^{R} \rho_{S,jT/R,b_2} \frac{T}{R} D_S^{-1/2}$ with error bounded by $O\big(\gamma(||b_1|| + ||x_S||)\big)$. From Lemma 7, this can be approximated to within $O(\gamma||x_S||)$ multiplicative error using $r = \gamma^{-2}\log(\sigma\gamma^{-1})$ samples with probability at least $1-\gamma$. This gives total additive and multiplicative error within $O(\gamma)$. □

We present an algorithm for computing a local solution to a Laplacian linear system with a boundary condition.

Theorem 8. *Let G be a graph and $\mathcal{L}$ denote the normalized Laplacian of G. Let b be a nontrivial vector $b \in \mathbb{R}^n$, S a b-boundable subset, and let $b_1 = D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$. For the linear system $\mathcal{L}x = b$, the solution x is required to satisfy the boundary condition b, and let x_S be the local solution. Then the approximate solution* $\mathtt{x}$ *output by the* LocalLinearSolver *algorithm has an error bounded by*

$$||x_S - \mathtt{x}|| = O\left(\gamma(||b_1|| + ||x_S||)\right)$$

with probability at least $1-\gamma$.

Proof. The correctness of the algorithm follows from Corollary 3. □

Algorithm 4. LocalLinearSolver(G, b, S, γ)

input: graph G, boundary vector $b \in \mathbb{R}^n$, subset $S \subset V$, solver error parameter $0 < \gamma < 1$.

output: an approximate local solution $\mathtt{x}$ with additive and multiplicative error γ to the local system $x_S = \mathcal{G} b_1$ satisyfing the boundary condition b.

$\sigma \leftarrow |S|$
initialize a 0-vector $\mathtt{x}$ of dimension σ
$b_1 \leftarrow D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$
$b_2 \leftarrow b_1^T D_S^{1/2}$
$T \leftarrow \sigma^3 \log(\sigma^3 \gamma^{-1})$
$R \leftarrow T/\gamma$
$r \leftarrow \gamma^{-2} \log(\sigma \gamma^{-1})$
for $i = 1$ to r **do**
    draw j from $[1, R]$ uniformly at random
    $x_i \leftarrow \rho_{S,jT/R,b_2}$
    $\mathtt{x} \leftarrow \mathtt{x} + x_i$
end for
return $T/r \cdot \mathtt{x} D_S^{-1/2}$

The algorithm involves $r = \gamma^{-2} \log(\sigma \gamma^{-1})$ Dirichlet heat kernel pagerank computations, so the running time is proportional to the time for computing $b_2 e^{-T(I_S - P_S)}$ for $T = \sigma^3 \log(\sigma^3 \gamma^{-1})$.

In the next sections, we discuss an efficient way to approximate a Dirichlet heat kernel pagerank vector and the resulting algorithm `GreensSolver` that returns approximate local solutions in sublinear time.

4.5 Computing Dirichlet Heat Kernel Pagerank

The definition of Dirichlet heat kernel pagerank in (4.8) is given in terms of a subset S and a vector $s \in \mathbb{R}^\sigma$. Our goal is to express this vector as the stationary distribution of random walks on the graph in order to design an efficient approximation algorithm.

Dirichlet heat kernel pagerank is defined over the vertices of a subset S as

follows:

$$\rho_{S,t,s} = s^T H_{S,t} = s^T e^{-t\Delta_S} = s^T e^{-t(I_S - P_S)}$$
$$= \sum_{k=0}^{\infty} e^{-t} \frac{t^k}{k!} f^T P_S^k.$$

That is, it is defined in terms of the transition probability matrix P_S – the restriction of P where P describes a random walk on the graph. We can interpret the matrix P_S as the transition probability matrix of the following so-called *Dirichlet random walk*: Move from a vertex v_i in S to a neighbor v_j with probability $1/d_i$. If v_j is not in S, abort the walk and ignore any probability movement. Since we only consider the diffusion of probability within the subset, any random walks which leave S cannot be allowed to return any probability to S. To prevent this, random walks that do not remain in S are ignored.

Similar to the general heat kernel pagerank, consider a Dirichlet random walk process in which the number of steps, k, are taken with probability $p_t(k) = e^{-t}\frac{t^k}{k!}$. Then, the Dirichlet heat kernel pagerank is the expected distribution of this process.

In order to use random walks for approximating Dirichlet heat kernel pagerank, we perform some preprocessing for general vectors $s \in \mathbb{R}^\sigma$. Namely, we do separate computations for the positive and negative parts of the vector, and normalize each part to be a probability distribution. Given a graph and a vector $s \in \mathbb{R}^\sigma$, the algorithm `ApproxDirHKPR` computes vectors that ϵ-approximate the Dirichlet heat kernel pagerank $\rho_{S,t,s}$ satisfying the criteria of Definition 2. When s is a general vector, an ϵ-approximate Dirichlet heat kernel pagerank vector has an additional additive error of $\epsilon||s||_1$ by scaling, where $||\cdot||_1$ denotes the L_1 norm.

The time complexity of `ApproxDirHKPR` is again given in terms of random walk steps and the analysis assumes access to constant-time queries returning the

Algorithm 5. `ApproxDirHKPR`(G, t, s, S, ϵ)

input: a graph G, $t \in \mathbb{R}^+$, vector $s \in \mathbb{R}^\sigma$, subset $S \subset V$, error parameter $0 < \epsilon < 1$.
output: ρ, an ϵ-approximation of $\rho_{S,t,s}$.

$\sigma \leftarrow |S|$
initialize 0-vector ρ of dimension σ
$s_+ \leftarrow$ the positive portion of s
$s_- \leftarrow$ the negative portion of s so that $s = s_+ - s_-$
$s'_+ \leftarrow s_+/||s_+||_1$ ▷ *normalize s_+ to be a probability distribution vector*
$s'_- \leftarrow s_-/||s_-||_1$ ▷ *normalize s_- to be a probability distribution vector*
$r \leftarrow \frac{16}{\epsilon^3} \log n$
for r iterations **do**
    choose a starting vertex v_1 according to the distribution vector s'_+
    choose k with probability $e^{-t}\frac{t^k}{k!}$
    $k \leftarrow min\{k, t/\epsilon\}$
    simulate k steps of a $P = D^{-1}A$ random walk
    if the random walk leaves S **then**:
        do nothing for the rest of this iteration
    else
        let v'_1 be the last vertex visited in the walk
        $\rho(v'_1) \leftarrow \rho(v'_1) + ||s_+||_1$
    end if
    choose a starting vertex v_2 according to the distribution vector s'_-
    choose k with probability $e^{-t}\frac{t^k}{k!}$
    $k \leftarrow min\{k, t/\epsilon\}$
    simulate k steps of a $P = D^{-1}A$ random walk
    if the random walk leaves S **then**:
        do nothing for the rest of this iteration
    else
        let v'_2 be the last vertex visited in the walk
        $\rho(v'_2) \leftarrow \rho(v'_2) + ||s_-||_1$
    end if
end for

$\rho \leftarrow \rho/r$
return ρ

destination of a random walk step, and a sample from a distribution.

Theorem 9. *Let G be a graph and S a proper vertex subset such that the induced subgraph on S is connected. Let s be a vector $s \in \mathbb{R}^{\sigma}$, $t \in \mathbb{R}^{+}$, and $0 < \epsilon < 1$. Then the algorithm* `ApproxDirHKPR`(G, t, s, S, ϵ) *outputs an ϵ-approximate Dirichlet heat kernel pagerank vector $\hat{\rho}_{S,t,s}$ with probability at least $1 - \epsilon$. The running time of* `ApproxDirHKPR` *is $O\left(\epsilon^{-4} t \log n\right)$, where the constant hidden in the big-O notation reflects the time to perform a random walk step.*

Proof. For the sake of simplicity, we provide analysis for the positive part of the vector, $s := s_{+}$, noting that it is easily applied similarly to the negative part as well.

The vector $s' = s/||s||_1$ is a probability distribution and the heat kernel pagerank $\rho'_{S,t,s} = \rho_{S,t,s}/||s||_1$ can be interpreted as a series of Dirichlet random walks in which, with probability $e^{-t}\frac{t^k}{k!}$, $s'^T P_S^k$ is contributed to $\rho'_{S,t,s}$. The probability of taking k steps such that $k \geq t/\epsilon$ is less than ϵ by Markov's inequality. Therefore, enforcing an upper bound of $K = t/\epsilon$ for the number of random walk steps taken is enough mixing time with probability at least $1 - \epsilon$.

For $k \leq t/\epsilon$, our algorithm approximates $s'^T P_S^k$ by simulating k random walk steps according to P as long as the random walk remains in S. If the random walk ever leaves S, it is ignored. Then, for random walks remaining in S, the analysis is identical to the proof of Theorem 1.

When s is not a probability distribution, the above applies to $s' = s/||s||_1$. Let $\hat{\rho}'_{S,t,s}$ be the output of the algorithm using $s' = s/||s||_1$ and $\rho'_{S,t,s}$ be the corresponding Dirichlet heat kernel pagerank vector $\rho_{S,t,s'}$. The full error of the

Dirichlet heat kernel pagerank returned is

$$\begin{aligned}
||\rho_{S,t,s} - \hat{\rho}_{S,t,s}||_1 &\leq ||\ ||s||_1\rho'_{S,t,s} - ||s||_1\hat{\rho}'_{S,t,s}\ ||_1 \\
&\leq ||s||_1||\rho'_{S,t,s} - \hat{\rho}'_{S,t,s}||_1 \\
&\leq \epsilon||s||_1||\rho'_{S,t,s}||_1 \\
&= \epsilon||s||_1.
\end{aligned}$$

For the running time, we use the assumptions that performing a random walk step and drawing from a distribution with finite support require constant time. These are incorporated in the random walk simulation, which dominates the computation. Therefore, for each of the r rounds, at most K steps of the random walk are simulated, giving a total of $rK = O\Big(\frac{16}{\epsilon^3}\log n \cdot t/\epsilon\Big) = \tilde{O}(t)$ queries. □

4.6 Computing Local Solutions With Random Walks

Here we present the main algorithm, `GreensSolver`, for computing a solution to a Laplacian linear system with a boundary condition. It is the LocalLinearSolver algorithmic framework combined with the scheme for approximating Dirichlet heat kernel pagerank. The scheme is an optimized version of the algorithm `ApproxDirHKPR` with a slight modification. We call the optimized version `SolverApproxDirHKPR`.

Definition 4. Define `SolverApproxDirHKPR`(G, t, s, S, ϵ) to be the algorithm `ApproxDirHKPR`(G, t, s, S, ϵ) with the following modification to lines 11 and 21:

$k \leftarrow min\{k, 2t\}$.

Namely, this modification limits the length of random walk steps to at most $2t$.

Theorem 10. *Let G be a graph and S a subset of size σ. Let $T = \sigma^3 \log(\sigma^3 \gamma^{-1})$, and let $R = T/\gamma$ for some $0 < \gamma < 1$. Suppose j is a random variable drawn from $[1, \lfloor R \rfloor]$ uniformly at random and let $t = jT/R$. Then if $\epsilon \geq \gamma$, the algorithm* `SolverApproxDirHKPR` *returns a vector that ϵ-approximates $\rho_{S,t,s}$ with probability at least $1 - \epsilon$. Using the same query assumptions as Theorem 9, the running time of* `SolverApproxDirHKPR` *is $O\left(\epsilon^{-3} t \log n\right)$.*

We will use the following Chernoff bound for Poisson random variables.

Lemma 8 ([67])**.** *Let X be a Poisson random variable with parameter t. Then, if $x > t$,*

$$\Pr(X \geq x) \leq e^{x - t - x \log(x/t)}.$$

Proof of Theorem 10. Let k be a Poisson random variable with parameter t. Similar to the proof of Theorem 9, we use Lemma 8 to reason that

$$\begin{aligned} \Pr(k \geq 2t) &\leq e^{2t - t - 2t \log(2t/t)} \\ &= e^{t(1 - 2\log 2)} \\ &\leq \epsilon, \end{aligned}$$

as long as $t \geq \frac{\log(\epsilon^{-1})}{1 - 2\log 2}$.

Let E be the event that $t < \frac{\log(\epsilon^{-1})}{1 - 2\log 2}$. The probability of E is

$$\begin{aligned} \Pr\left(jT/R < \frac{\log(\epsilon^{-1})}{1 - 2\log 2}\right) &= \Pr\left(j < \frac{\log(\epsilon^{-1})}{\gamma(1 - 2\log 2)}\right) \\ &= \frac{\log(\epsilon^{-1})}{(1 - 2\log 2)\sigma^3 \log(\sigma^3 \gamma^{-1})}, \end{aligned}$$

which is less than ϵ as long as $\epsilon \geq \left(\frac{\gamma}{\sigma^3}\right)^{(1 - 2\log 2)\epsilon\sigma^3}$. This holds when $\epsilon \geq \gamma$.

As before, the algorithm consists of r rounds of random walk simulation,

where each walk is at most $2t$. The algorithm therefore makes $r \cdot 2t = \epsilon^{-3}32t \log n$ queries, requiring $O(\epsilon^{-3}t \log n)$ time. □

Below we give the algorithm `GreensSolver`. The algorithm is identical to LocalLinearSolver with the exception of line 10, where we use the approximation algorithm `SolverApproxDirHKPR` for Dirichlet heat kernel pagerank computation.

Algorithm 6. `GreensSolver`$(G, b, S, \gamma, \epsilon)$

input: graph G, boundary vector $b \in \mathbb{R}^n$, subset $S \subset V$, solver error parameter $0 < \gamma < 1$, Dirichlet heat kernel pagerank error parameter $0 < \epsilon < 1$.
output: an approximate local solution $\mathtt{x}$ to the local system $x_S = \mathcal{G}b_1$ satisyfing the boundary condition b.

1: $\sigma \leftarrow |S|$
2: initialize a 0-vector $\mathtt{x}$ of dimension σ
3: $b_1 \leftarrow D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$
4: $b_2 \leftarrow b_1^T D_S^{1/2}$
5: $T \leftarrow \sigma^3 \log(\sigma^3 \gamma^{-1})$
6: $R \leftarrow T/\gamma$
7: $r \leftarrow \gamma^{-2} \log(\sigma\gamma^{-1})$
8: **for** $i = 1$ to r **do**
9: draw j from $[1, R]$ uniformly at random
10: $x_i \leftarrow \mathtt{SolverApproxDirHKPR}(G, jT/R, b_2, S, \epsilon)$
11: $\mathtt{x} \leftarrow \mathtt{x} + x_i$
12: **end for**
13: **return** $T/r \cdot \mathtt{x} D_S^{-1/2}$

Theorem 11. *Let G be a graph and $\mathcal{L}$ denote the normalized Laplacian of G. Let b be a nontrivial vector $b \in \mathbb{R}^n$ and S a b-boundable subset, and let $b_1 = D_S^{-1/2} A_{S,\delta S} D_{\delta S}^{-1/2} b_{\delta S}$. For the linear system $\mathcal{L}x = b$, the solution x is required to satisfy the boundary condition b, and let x_S be the local solution. Then the approximate solution $\mathtt{x}$ output by the algorithm `GreensSolver` satisfies the following:*

(i) The error of $\mathtt{x}$ is $||x_S - \mathtt{x}|| = O(\gamma(||b_1|| + ||x_S||) + \epsilon||b_2||_1)$ with probability at least $1 - \gamma$,

(ii) The running time of ***GreensSolver*** *is* $O\left(\gamma^{-2}\epsilon^{-3}\sigma^3\log^2(\sigma^3\gamma^{-1})\log n\right)$ *where the big-O constant reflects the time to perform a random walk step, plus additional preprocessing time* $O(|\partial(S)|)$*, where* $\partial(S)$ *denotes the edge boundary of* S*.*

Proof. The error of the algorithm using true Dirichlet heat kernel pagerank vectors is $O\big(\gamma(||b_1||+||x_S||)\big)$ by Corollary 3, so to prove (i) we address the additional error of vectors output by the approximation of `SolverApproxDirHKPR`. By Theorem 10, `SolverApproxDirHKPR` outputs an ϵ-approximate Dirichlet heat kernel pagerank vector with probability at least $1-\epsilon$. Let $\hat{\rho}_{S,t,s}$ be the output of an arbitrary run of `SolverApproxDirHKPR`(G,t,s,S,ϵ). Then $||\rho_{S,t,s}-\hat{\rho}_{S,t,s}|| \leq \epsilon(||\rho_{S,t,s'}||_1+||s||_1) = \epsilon||s||_1$ by the definition of ϵ-approximate Dirichlet heat kernel pagerank vectors, where $s'=s/||s||_1$ is the normalized vector s. This means that the total error of `GreensSolver` is

$$||x_S-\mathtt{x}|| \leq O\left(\gamma(||b_1||+||x_S||)\right)+\epsilon||b_2||_1.$$

Next we prove (ii). The algorithm makes $r=\gamma^{-2}\log(\sigma\gamma^{-1})$ sequential calls to `SolverApproxDirHKPR`. The maximum possible value of t is $T=\sigma^3\log(\sigma^3\gamma^{-1})$, so any call to `SolverApproxDirHKPR` is bounded by $O\left(\epsilon^{-3}\sigma^3\log(\sigma^3\gamma^{-1})\log n\right)$. Thus, the total running time is $O\left(\gamma^{-2}\epsilon^{-3}\sigma^3\log^2(\sigma^3\gamma^{-1})\log n\right)$.

The additional preprocessing time of $O(|\partial(S)|)$ is for computing the vectors b_1 and b_2; these may be computed as a preliminary procedure. □

We note that the running time above is a sequential running time attained by calling `SolverApproxDirHKPR` r times. However, by calling these in r parallel processes, the algorithm has a parallel running time which is simply the same as that for `SolverApproxDirHKPR`. Since `SolverApproxDirHKPR` only promises

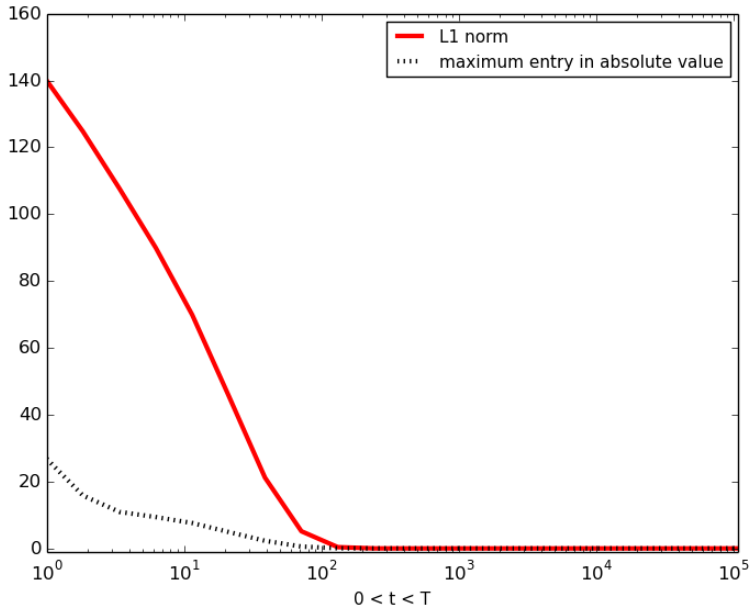

Figure 4.2. Support values of a Dirichlet heat kernel pagerank vector different values of t. The solid line is the L_1 norm and the dashed line is the absolute value of the maximum entry in the vector. Note the x-axis is log-scale.

approximate values for vertices whose true Dirichlet heat kernel pagerank vector values are greater than ϵ, the

`GreensSolver` algorithm can be optimized even further by preempting when this is the case.

Figure 4.2 illustrates how vector values drop as t gets large. The network is again the dolphins social network and is further examined in the next section. We let t range from 1 to $T = \sigma^3 \log(\sigma^3 \gamma^{-1}) \approx 108739$ for $\gamma = 0.01$ and compute Dirichlet heat kernel pagerank vectors $\rho_{S,t,s}$. The figure plots L_1 norms of the vectors as a solid line, and the absolute value of the maximum entry in the vector as a dashed line. In this example, no vector entry is larger than 0.01 for t as small as 250.

Suppose it is possible to know ahead of time whether a vector $\rho_{S,t,s}$ will have

negligably small values for some value t. Then we could skip the computation of this vector and simply treat it as a vector of all zeros.

From (4.7), the norm of Dirichlet heat kernel pagerank vectors are monotone decreasing. Then it is enough to choose a threshold value t' beyond which $||\rho_{S,t',s}||_1 < \epsilon$, since any ϵ-approximation will return all zeros, and treat this as a cutoff for actually executing the algorithm. An optimization heuristic is to only compute `SolverApproxDirHKPR`(G, t, s, S, ϵ) if t is less than this threshold value t'. Otherwise we can add zeros (or do nothing). That is, replace line 10 in `GreensSolver` with the following:

if $jT/N < t'$ **then**

$x_i \leftarrow$ `SolverApproxDirHKPR`$(G, jT/N, b_2, S, \epsilon)$

else

do nothing

end if

From (4.7), a conservative choice for t' is $\frac{1}{\lambda_S} \log(\epsilon^{-1})$.

4.7 An Example Illustrating the Algorithm

We return to our example to illustrate a run of the Green's solver algorithm for computing local linear solutions. The network is a small communication network of dolphins [64].

In this example, the subset has a good cluster, which makes it a good candidate for an algorithm in which computations are localized. Namely, it is ideal for `SolverApproxDirHKPR`, which promises good approximation for vertices that exceed a certain support threshold in terms of the error parameter ϵ. The support of the vector b is limited to the set of leaders, which is the vertex boundary of the

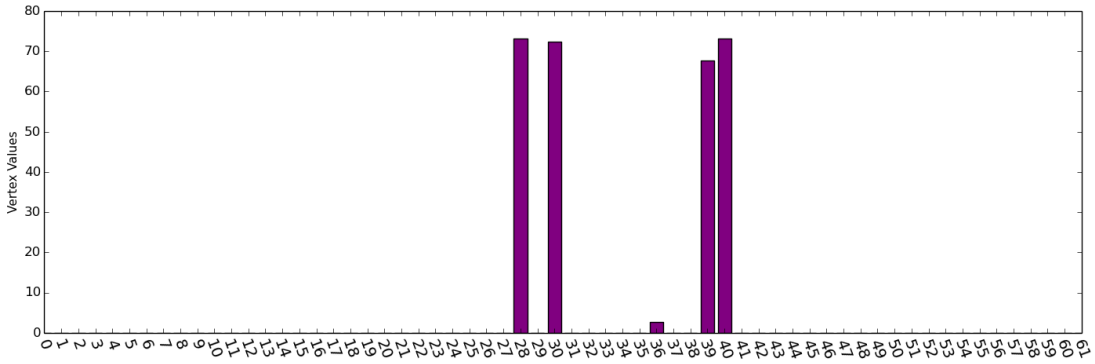

Figure 4.3. The values of the boundary vector plotted against the agent IDs given in Figure 4.1.

subset of followers, $l = \delta(f)$. The vector is plotted over the agents (vertices) in Figure 4.3.

Figure 4.4 plots the vector values of the heat kernel pagerank vector $\rho_{t,b_2'}$ over the full set of agents. Here, we use b_2', the n-dimensional vector:

$$b_2'(v) = \begin{cases} b_2(v) \text{ if } v \in S, \\ 0 \text{ otherwise,} \end{cases}$$

and $t = 50.0$. The components with largest absolute value are concentrated in the subset of followers over which we compute the local solution. This indicates that an output of `SolverApproxDirHKPR` will capture these values well.

In the following figures, we plot the results of calls to our approximation algorithms against the exact solution x_S using the boundary vector of Figure 4.3. The solution x_S is computed by Theorem 5, and the appromimations are sample outputs of LocalLinearSolverand `GreensSolver`, respectively. The exact values of x_S are represented by circles, and the approximate values by triangles in each case. Note that we permute the indices of the vertices in the solutions so that

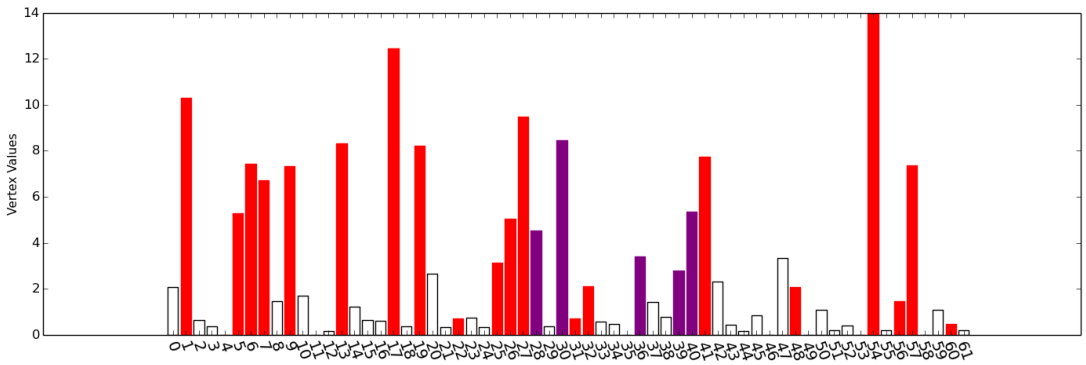

Figure 4.4. The vertex values of the full example communication network over a sample heat kernel pagerank vector. The red bars correspond to the network of followers, the purple to the leaders, and the white to the rest of the network.

vector values in the exact solution, x_S are decreasing, for reading ease.[1]

The result of a sample call to LocalLinearSolver with error parameter $\gamma = 0.01$ is plotted in Figure 4.5. The total relative error of this solution is $\frac{||x_S - \hat{x}_S||}{||x_S||} = 0.02$, and the absolute error $||x_S - \hat{x}_S||$ is within the error bounds given in Theorem 8. That is, $||x_S - \hat{x}_S|| \leq \gamma\left(||b_1|| + ||x_S|| + ||x_{rie}||\right)$, where x_{rie} is the solution obtained by computing the full Riemann sum (as in Lemma 6).

The result of a sample call to `GreensSolver` with parameters $\gamma = 0.01, \epsilon = 0.1$ is plotted in Figure 4.6. In this case the relative error is ≈ 2.05, but the absolute error meets the error bounds promised in Theorem 11 point (i). Specifically,

$$||x_S - \hat{x}_S|| \leq \left(\gamma(||b_1|| + ||x_S|| + ||x_{rie}||) + \epsilon||b_2||_1\right).$$

General remarks. While we have focused our analysis on solving local linear systems with the normalized Laplacian $\mathcal{L}$ as the coefficient matrix, our methods can

[1]The results of these experiments as well as the source code are archived at http://cseweb.ucsd.edu/~osimpson/localsolverexample.html.

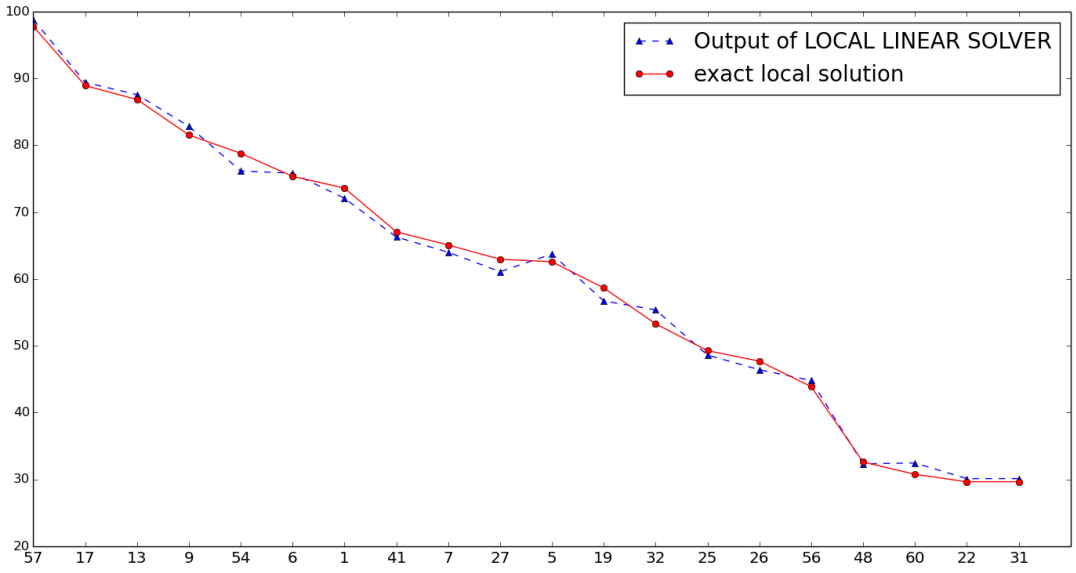

Figure 4.5. The results of a run of LocalLinearSolver. Two vectors are plotted over IDs of agents in the subset. The circles are exact values of x_S, while the triangles are the approximate values returned by LocalLinearSolver.

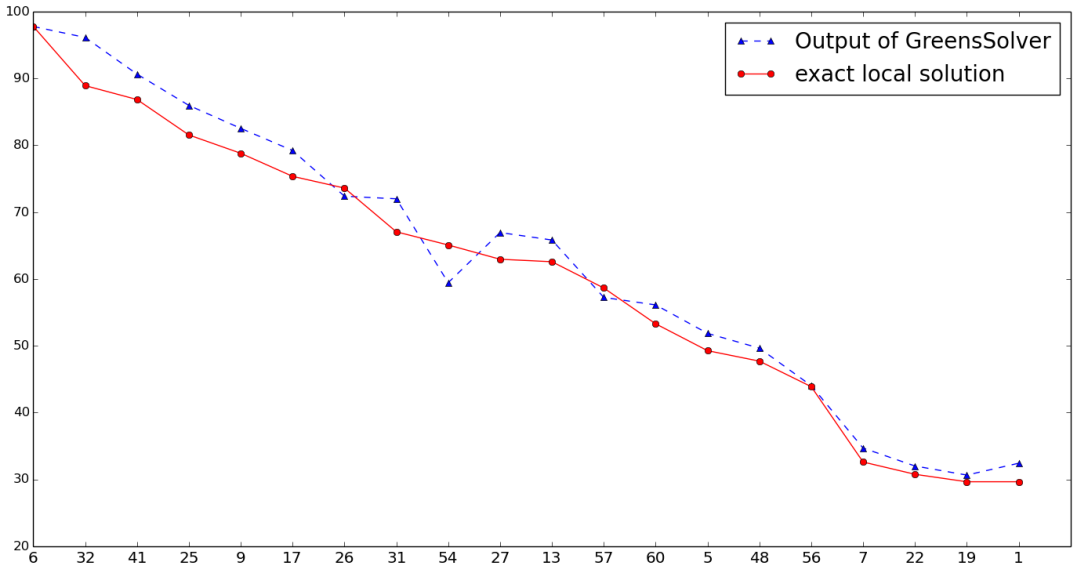

Figure 4.6. The results of a run of `GreensSolver` with $\gamma = 0.01, \epsilon = 0.1$.

be extended to solve local linear systems expressed in terms of the Laplacian L as well. There are numerous applications involving solving such linear systems. Some examples are discussed in [25], and include computing effective resistance in electrical networks, computing maximum flow by interior point methods, describing the motion of coupled oscillators, and computing state in a network of communicating agents. In addition, we expect the method of approximating Dirichlet heat kernel pagerank in its own right to be useful in a variety of related applications.

Acknowledgements

Chapter 4, in part, is a reprint of the material as it appears in *Internet Mathematics*. Fan Chung and Olivia Simpson, vol 11(4-5), 2015 pp. 449-471. The dissertation author was a primary investigator and author of this paper.

Chapter 5

Computing Consensus

The problem of consensus among multi-agent systems has wide applications in situations where members of a distributed network must agree. For example, the communication, feedback, and decision-making between distinct unmanned aerial vehicles (UAVs) [35, 73] is closely related to the consensus problem. In addition to UAVs, the distributed coordination of networks has important implications in cooperative control of distributed sensor networks [27], flocking and swarming behavior [91], and communication congestion control [78]. Further, they form the foundation of the field of distributed computing [65]. The consensus problem is studied in [74], and several variations and extensions are examined by [20, 68, 71, 75].

We consider the classical model (see [74]) of agents with fixed, bidirectional communication channels and associated state. State changes occur continuously, influenced by communication with neighbors. A consensus algorithm is a continuous time protocol that specifies the information exchange between agents and provides a mechanism for systematically computing the consensus value, a unanimous state and an equilibrium of the system. In this chapter, we focus on an efficient method for approximating the state values in a network in which agents reach consensus by following a linear protocol. We give algorithms for two different frameworks. The first computes a global consensus value involving all the agents in the network and

runs in time sublinear in the size of the network. The second is a local algorithm to compute a consensus value for a subset of agents under external influence, and runs in time sublinear in the size of the specified subset. In both, the consensus value returned is within an error bound of $O(\epsilon)$ for a given $0 < \epsilon < 1$.

The algorithms presented in this chapter are extension of the results Chapter 4 for solving a linear system by approximating the heat kernel pagerank of the network for the purpose of consensus.

5.1 Previous Work on Computing Consensus

In [74], Olfati-Saber and Murray design a linear protocol for agents to reach a consensus value which is an average of initial states. They consider a network of agents as an undirected graph and use the Laplacian potential, defined in terms of the graph Laplacian, as a measure of disagreement among vertices. With this tool, they transform the the problem of reaching consensus to that of minimizing the Laplacian potential.

An alternate formulation given in [35] abides by a linear protocol which favors the values of more highly connected vertices. In this way, agents which are more visible will have more of an impact on the group decision. Yet another variation is consensus in a leader-following formation, in which a set of agents called leaders abide by individual protocol but continue to influence to rest of the network. This problem has been studied in [71, 82].

In the model we consider, the communication protocol followed by the agents in the network forms a linear system of equations, and the solution to the linear system is the state of the network as a function of time. Thus, computing the consensus value involves solving a linear system.

5.2 Multi-Agent Systems

A dynamic multi-agent system is given by a tuple $G_x = (G, x)$ where x is the state of the system and G is the communication network topology, represented by a graph. Namely, each agent is represented by a vertex and the communication network between agents is represented by the edge set E. Let $x_i \in \mathbb{R}$ be a real scalar value assigned to vertex v_i such that $x(t) = (x_1(t), \ldots, x_n(t))^T$ denotes the state at time t.

Two vertices v_i, v_j are said to *agree* if and only if $x_i = x_j$. The goal of consensus is to minimize the total disagreement among vertices.

Definition 5 (Consensus)**.** Let the value of vertices x be the solution to the equation

$$\frac{\partial u}{\partial t} = f(x, u),\ x(0) \in \mathbb{R}^n. \tag{5.1}$$

Let $X : \mathbb{R}^n \to \mathbb{R}$ be an operator on $x = (x_1, \ldots, x_n)^T$ that generates a decision value $X(x)$. Then we say all vertices of the graph have reached *consensus with respect to* X in finite time $T > 0$ if and only if all vertices agree and $x_i(T) = X(x(0))\ \forall\ i \in \mathcal{I}$. We call $X(x) := X(x(0))$ the *consensus value*.

One notion of consensus is a *weighted average consensus*, given by

$$X_w(x) = \frac{\sum_i d_i x_i}{\sum_i d_i}.$$

We show (Theorem 12) that any connected undirected graph globally asymptotically reaches weighted average consensus when each vertex applies the distributed linear protocol

$$u_i(t) = 1/d_i \sum_{j \in N_i} (x_j(t) - x_i(t)). \tag{5.2}$$

Again, we assume G is connected.

5.3 Heat Kernel Pagerank for the Weighted-Average Consensus Problem

In this section we present a linear consensus protocol for a dynamic network and show how to compute a weighted average consensus for the protocol using heat kernel pagerank.

We first recall some principles of control theory. Consider the system with controls as in (5.1). A point x_e is an *equilibrium point* of the system if $f(x_e, u) = 0$, and x_e is an equilibrium point if and only if $x(t) = x_e$ is a trajectory. The system is *globally asymptotically stable* if, for every trajectory $x(t)$, $x(t) \to x_e$ as $t \to \infty$. To check this, two conditions are sufficient.

Definition 6 (Global asymptotic stability)**.** A system is *globally asymptotically stable* if

1. it is stable in the Lyapunov sense, and
2. the equilibrium x_e is convergent, i.e., for every $\epsilon > 0$, there is some time T such that

$$||x(0) - x_e|| < \delta \quad \text{means} \quad ||x(t) - x_e|| < \epsilon$$

for every time $t > T$.

In particular, when considering a time-invariant linear state space model $\dot{x} = -Mx$, for some matrix M, condition 1 is satisfied if M is positive semidefinite.

Consider the network of integrator agents with dynamics $\frac{\partial x}{\partial t} = u_i(t)$ where each agent applies the distributed linear protocol (5.2). We can characterize the

dynamics of the system by the Laplace operator $\Delta = I - P$ for the underlying graph, as described by the following theorem:

Theorem 12. *Let G_x be a dynamic multi-agent system and suppose each vertex of G applies the distributed linear protocol (5.2). Then the value of x at time t is given by the solution to the system*

$$\frac{\partial x}{\partial t} = -\Delta x(t), \quad x(0) \in \mathbb{R}^n. \tag{5.3}$$

Additionally, this protocol globally asymptotically reaches a weighted average consensus.

Proof. Let x_e be an equilibrium of the system $\frac{\partial x}{\partial t} = -\Delta x$. Then by definition of equilibrium, $\Delta x_e = 0$ and therefore x_e is a right eigenvector associated to the eigenvalue $\lambda = 0$. In particular x_e is in the null space of Δ. Since G is connected, Δ has exactly one zero eigenvalue. Upon consideration, we see that the corresponding eigenvector is $\mathbf{1}$, the all-one's vector, as the row sums of Δ are all exactly zero. Thus, $x_e = \alpha\mathbf{1}$ for some $\alpha \in \mathbb{R}$. Now, note that $\sum_i u_i = 0$ for the protocol (5.2), and so the weighted average value $X_w(x(t))$, determined by $u(t)$, is in fact invariant with respect to t. In other words, $X_w(x(0)) = X_w(x_e)$, and

$$X_w(x_e) = \frac{\sum_i d_i(x_e)_i}{\sum_i d_i} = \alpha.$$

Therefore this equilibrium is in fact the weighted average of the initial values of the vertices, and all vertices reach this value. Also, as the system is time-invariant, the system is stable since Δ is positive semidefinite.

By Definition 6, the Theorem is proved. □

Now we can summarize the state of the system with a single heat kernel

pagerank vector.

Theorem 13. *Let G_x be a dynamic multi-agent system and suppose each vertex of G applies the distributed linear protocol (5.2). Let D be the diagonal degree matrix of G. Then the state of the system is given in terms of heat kernel pagerank by*

$$x(t) = \rho_{T,s} D^{-1}, \quad s^T = x(0)^T D. \tag{5.4}$$

Proof. The solution to (5.3) is the evolving state of the system. This solution is

$$x(t) = e^{-t\Delta} x(0) = H_t x(0). \tag{5.5}$$

Using the symmetrized version of heat kernel,

$$\begin{aligned} x(t) &= (D^{-1/2} \mathcal{H}_t D^{1/2}) x(0) \\ x(t)^T &= x(0)^T D^{1/2} (D^{1/2} H_t D^{-1/2}) D^{-1/2} \\ x(t)^T &= (x(0)^T D) H_t D^{-1}, \end{aligned} \tag{5.6}$$

where line 5.6 uses the symmetry of D and $\mathcal{H}$. Thus, the values $x(t)$ given by (5.5) are related to the heat kernel pagerank vector $\rho_{t,s}$ with preference vector $s^T = x(0)^T D$. □

To compute the equilibrium state at which all agents reach consensus, we know that time $T = O(1/\lambda_1)$ is an upper bound. Figure 5.1 depicts the results of computing weighted average consensus with heat kernel pagerank as in Theorem 13 with different values for t. The network is the dolphins network [64] with initial state values randomly chosen from the interval $(0, 1)$. The chart plots total disagreement $||\delta||$ for disagreement vector $\delta(t) = x(t) - \chi_w(x)\mathbf{1}$, where $x(t) = \rho_{t,s} D^{-1}$ for

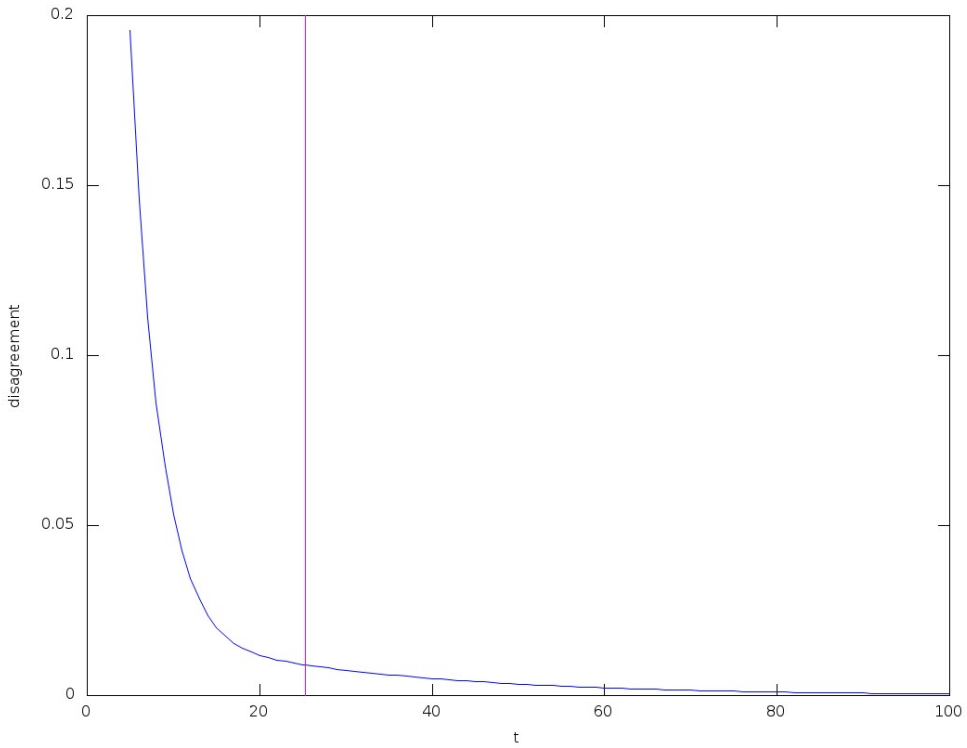

Figure 5.1. Total disagreement over varying times t. Disagreement is computed in terms of the weighted average consensus $\chi_w(x_0)$. The red line denotes the data point for $t = 1/\lambda_1$.

$f = x(0)^{tr}D$. The vertical line corresponds to $t = 1/\lambda_1$.

Our weighted average consensus algorithm uses an algorithm for computing an ϵ-approximate heat kernel pagerank as defined in Definition 2 as a subroutine. As mentioned in Chapter 2, a general heat kernel pagerank vector can be computed as a combination of "seeded" heat kernel pagerank vectors. As such, we can simply adapt the `ApproxHKPRseed` to compute a general heat kernel pagerank vector by allowing each random walk to start from a vertex drawn from s. We will refer to this algorithm `ApproxHKPR`. We call our algorithm for consensus AvgConsensus.

Theorem 14 (Weighted Average Consensus in Sublinear Time)**.** *Let G_x be a*

dynamic n-agent system and suppose each vertex of G applies the distributed linear protocol (5.2). Then the state of the system can be approximated to within a multiplicative factor of $(1+\epsilon)$ and an additive term of $O(\epsilon)$ for any $0 < \epsilon < 1$ in time $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$.

Proof. First, the ϵ-approximate vector x returned by `ApproxHKPR` is an approximation of the true state by Theorem 13. Thus we have left to verify the approximation guarantee and the running time. The total running time is dominated by the heat kernel pagerank approximation, which is $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3 \log\log(\epsilon^{-1})}\right)$ by Theorem 1. □

Algorithm 7. AvgConsensus(G, x, t, ϵ)

input: graph G, initial state vector x, $t \in \mathbb{R}^+$, error parameter $0 < \epsilon < 1$.
output: an approximate state $x(t)$.

1: $D \leftarrow$ diagonal matrix of vertex degrees
2: $s^T \leftarrow x^T D$
3: $y \leftarrow$ `ApproxHKPR`(G, t, s, ϵ)
4: **return** yD^{-1}

5.4 Heat Kernel Pagerank for Consensus in Leader-Following Formations

In this section we consider a multi-agent network in which a certain subset of agents $l \subset V$ are *leaders*, and the rest $f = V \setminus l$ are dubbed *followers*. In this scenario, leaders will adjust their values according to individual protocol, while followers in the system adjust according to communication channels as usual. The consensus goal in this case is a *leader-following consensus*, in which all agents agree on a value by following the leaders.

Let u^f denote the protocol among the set of followers and let u^l denote the control dictated by the leaders and influencing the followers. Similarly, let x^f

denote the state of the followers and x^l denote the state of the leaders. The vectors x^f and x^l can be understood as the usual state vector x restricted to following and leading agents, respectively. Then we have the following definition.

Definition 7 (Leader-following consensus)**.** A *leader-following consensus* of a system is achieved if for every agent v_i there is a local protocol u_i such that $x_i(T) = X_{lf}(x(0))$ for some finite time $T > 0$ and some operator $X_{lf} : \mathbb{R}^n \to \mathbb{R}$. In this case, we call the value $X_{lf}(x(0))$ the leader-following consensus value.

For the protocol

$$u_i(t) = 1/d_i \sum_{j \in N_i} \left(\sqrt{\frac{d_i}{d_j}} x_j(t) - x_i(t) \right), \tag{5.7}$$

the value for x is given by the dynamics

$$u(t) = \frac{\partial x}{\partial t} = -\mathcal{L}x(t).$$

We let the followers abide by protocol (5.7).

Let $\mathcal{L}_f$ be the Laplacian $\mathcal{L}$ restricted to rows and columns corresponding to the followers, and $\mathcal{L}_{fl}$ be $\mathcal{L}$ with rows restricted to the followers and columns restricted to the leaders. Then the dynamics of the followers can be summarized by:

$$\frac{\partial x^f}{\partial t} = -\mathcal{L}_f x^f(t) - \mathcal{L}_{fl} u^l(t).$$

Since $\frac{\partial x^f}{\partial t}$ is control of the subnetwork induced by the group of followers, this can

be rewritten

$$u^f = -\mathcal{L}_f x^f - \mathcal{L}_{fl} u^l \qquad \text{or alternatively,}$$
$$x^f = -\mathcal{L}_f^{-1} u^f + \mathcal{L}_f^{-1}\mathcal{L}_{fl} u^l.$$

Indeed, as long as the subgraph induced by the subset of followers is connected, the inverse $\mathcal{L}_f^{-1}$ exists. We have arrived at the following.

Theorem 15. *Let G_x be a dynamic multi-agent system with proper subsets of leaders, $l \subset V$, and followers, $f = V \setminus l$, such that the induced subgraph on f is connected. Suppose the followers apply the protocol (5.7), and suppose the leaders apply some individual protocol $u_i = f(x_i)$ dictated only by that leader's state. Then the followers' state values x^f at time t are given by the solution to the system*

$$\mathcal{L}_f x^f(t) = b(t), \qquad \textit{where} \qquad b(t) = -(u^f(t) + \mathcal{L}_{fl} u^l(t)).$$

We use the algorithm `GreensSolver` for solving the linear system $\mathcal{L}_f x^f = b$ with a linear protocol applied to a subset f specified by b. The solution $\mathcal{L}_f^{-1} b$ can be approximated by sampling sufficiently many values of $(\mathcal{H}_t)_f b(t)$. Further, it is given that the solution can be approximated in $O\left(\gamma^{-2}\epsilon^{-3}\sigma^3 \log^2(\sigma^3\gamma^{-1}) \log n\right)$ time, where σ is the size of the subset of followers.

The running time and approximation guarantees of LFConsensus follow from the running time of `GreensSolver`, and we have the following:

Theorem 16. *Let G_x be a dynamic multi-agent system with proper subsets of leaders, $l \subset V$, and followers, $f = V \setminus l$, such that the induced subgraph on f is connected. Suppose the followers apply the protocol (5.7), and suppose the leaders apply some individual protocol $u_i = f(x_i)$ dictated only by that leader's state. Then the state*

Algorithm 8. LFConsensus$(G, x, t, f, l, u^l, \epsilon)$

input: graph G, initial state vector x, $t \in \mathbb{R}^+$, subset of followers f, subset of leaders l, protocol applied by the leaders u^l, error parameter $0 < \epsilon < 1$.
output: an approximate state $x(t)$.

$b \leftarrow \mathrm{B}(t)$
$s \leftarrow |f|$
$T \leftarrow \sigma^3 \log(\sigma^3 \gamma^{-1})$
$R \leftarrow T/\epsilon$
$r \leftarrow \epsilon^{-2} \log(\sigma \epsilon^{-1})$
initialize a 0-vector x^f of dimension σ
for $i = 1$ to r **do**
    draw j from $[1, R]$ uniformly at random
    $x_i \leftarrow \texttt{SolverApproxDirHKPR}(G, jT/N, b, f, \epsilon)$
    $x^f \leftarrow x^f + x_i$
end for
return $T/r \cdot \mathtt{x} D_S^{-1/2}$

Procedure 9. FollowerProt(G, x, t)

for $i \in l$ **do**
    $u^f(i) \leftarrow u_i(t) = 1/d_i \sum_{j \in N_i} \left(\sqrt{\frac{d_i}{d_j}} x_j(t) - x_i(t) \right)$
end for
return u^f

Procedure 10. B(t)

$u^f \leftarrow \text{FollowerProt}(G, x, t)$
$b \leftarrow -(u^f(t) + \mathcal{L}_{fl} u^l)$
return b

of the system can be approximated to within a multiplicative factor of $(1+\epsilon)$ *and an additive term of* $O(\epsilon)$ *for any* $0 < \epsilon < 1$ *in time* $O\left(\gamma^{-2}\epsilon^{-3}\sigma^3 \log^2(\sigma^3\gamma^{-1}) \log n\right)$*, where* σ *is the size of the subset of followers.*

The significance of sublinear running times is scalability. The robustness and efficiency of the algorithms AvgConsensus and LFConsensus are of great importance for networks too large to fit in memory, and the running time/approximation tradeoff allows for appropriate tuning. This is especially notable for local algorithms, which reduce computation over large networks to a small subset. For instance, while the group of leaders may be small in a leader-following framework, the difference in complexity for computing consensus in a leader-following formation as opposed to full group consensus can be significant. The subset of followers influenced by the leaders may be a small portion of the entire graph, so that $s << n$, and we are spared work over the entire graph in the case that we are interested in only a small area. In these cases, the gain in running times are valueable.

Chapter 6

Distributed Algorithms for Finding Local Clusters

Distributed computation is an increasingly important framework as the demand for fast data analysis grows and data simultaneously becomes too large to fit in main memory. As distributed systems for large-scale graph processing such as Pregel [66], GraphLab [63], and Google's MapReduce [32] are rapidly developing, there is a need for both theoretical and practical bounds in adapting classical graph algorithms to a modern distributed and parallel setting.

A distributed algorithm performs local computations on pieces of input and communicates the results through given communication links. When processing a massive graph in a distributed algorithm, local outputs must be configured without shared memory and with few rounds of communication. A central problem of interest is to compute local clusters in large graphs in a distributed setting (see Chapter 3 for background on local cluster detection in a centralized setting).

In this chapter, we present the first algorithms for computing local clusters in two distributed settings that finish in a sublinear number of rounds of communication. In the distributed context, we are able to exploit parallelism in our algorithm for computing the heat kernel pagerank and give a distributed random walk-based

procedure which requires fewer rounds of communication and yet maintains similar approximation guarantees as previous distributed algorithms for computing local clusters. We will describe two distributive models – the CONGEST model and the k-machine model. We demonstrate in two different distributed settings that a heat kernel pagerank distribution can be used to compute local clusters with Cheeger ratio $O(\sqrt{\phi})$ when the optimal local cluster has Cheeger ratio ϕ.

6.1 Previous Work on Distributed Algorithms

Modeling distributed computation has been an important paradigm since the "many server" model has grown in popularity. Das Sarma et al. [31] give fast random walk-based distributed algorithms for estimating mixing time, conductance and the spectral gap of a network. In [30], distributed algorithms are derived for computing PageRank vectors with $O(\frac{1}{\alpha}\log n)$ rounds for any $0 < \alpha < 1$ with high probability. Das Sarma et al. [29] give two algorithms for computing sparse cuts in the CONGEST distributed model. The first algorithm uses random walks and is based on the analysis of [88]. By incorporating the results of [31], they show that the stationary distribution of a random walk of length l can be computed in $O(l)$ rounds. The second algorithm in [29] uses PageRank vectors and is based on the analysis of [6]. By using the results of [30], the authors of [29] compute local clusters in $O((\frac{1}{\phi}+n)\log n)$ rounds with standard random walks and $O(\frac{1}{\alpha}\log^2 n + n\log n)$ rounds using PageRank vectors. Finally, Bahmani, Chowdhury, and Goel [10] give an analysis of the efficiency of Monte Carlo algorithms in estimating PageRank.

6.2 Models of Distributed Computation

We consider two models of distributed computation – the CONGEST model and the k-machine model. In each, data is distributed across machines of a network

which may communicate over specified communication links in rounds. Memory is decentralized, and the goal is to minimize the running time by minimizing the number of rounds required for computation for an arbitrary input graph G. We emphasize that local communication is taken to be free.

The first model we consider is the CONGEST model. In this model, the communication links are exactly the edges of the input graph and each vertex is mapped to a dedicated host machine. The CONGEST (or standard message-passing) model was introduced in [79, 80] to simulate real-world bandwidth restrictions across a network.

Due to how the vertices are distributed in the network, we simplify the model by assuming the computer network is the input graph $G = (V, E)$ on $n = |V|$ machines with $m = |E|$ communication links. Each machine has a unique $\log n$-bit ID. Initially each machine only possesses its own ID and the IDs of each of its neighbors, and in some instances we may allow machines some metadata about the graph (the value of n, for instance). Machines can only communicate through communication links of the network and communication occurs in rounds. That is, any message sent at the beginning of round r is fully transmitted and received by the end of round r. We assume that all machines run with the same processing speed. Most importantly, we only allow $O(\log n)$ bits to be transmitted across any link per round.

The defining difference between the k-machine model and the CONGEST model is that, whereas vertices are mapped to distinct, dedicated machines in the CONGEST model, a number of vertices may be mapped to the same machine in the k-machine model. This model is meant to more accurately simulate distributed graph computation in systems such as Pregel [66] and GraphLab [63].

We consider computing over massive datasets distributed over machines

of the k-machine network. The complete data is never known by any individual machine, and there is no shared memory. Each machine executes an instance of a distributed algorithm, and the output of each machine is with respect to the data it hosts. A solution to a full problem is then a particular configuration of the outputs of each of the machines. The model is discussed in greater detail in Section 6.5.

The two models are limiting and advantageous in different ways, and one is not inherently better than the other. For instance, since many vertices are mapped to a single machine in the k-machine model, there is more "local information" available since vertices sharing a machine can communicate for free. However, since communication is restricted to the communication links in the computer network, vertex-vertex communication is somewhat less restrictive in the CONGEST model since links exactly correspond to edges. The consequences of these differences are largely observed in time complexity, and certain graph problems are more suited to one model than the other.

In this chapter we analyze our algorithmic techniques in the CONGEST model, and then use the Conversion Theorem of [47] to give an efficient probabilistic algorithm in the k-machine model for computing local clusters.

6.3 Fast Distributed Heat Kernel Pagerank Computation

The idea of the algorithm is to launch a number of random walks from the machine hosting the seed vertex v (seed machine) in parallel and compute the fraction of random walks which end at a machine hosting vertex v_i as an estimate of the heat kernel pagerank values $\rho_{t,v}(v_i)$.

To be specific, the seed machine initializes r tokens, each of which holds a random variable k corresponding to the length of its random walk. Then, in

rounds, the tokens are passed to random neighbors with a count incrementor until the count reaches k. At the end of the parallel random walks, each machine holding tokens outputs the number of tokens it holds divided by r as an estimate for the heat kernel pagerank value of the vertex it hosts. Algorithm 11 describes the full procedure.

Algorithm 11. `DistributedApproxHKPRseed`(G, t, v, ϵ)

input: a graph G hosted by a network in the CONGEST model, a seed vertex v, a diffusion parameter t, an error bound ϵ
output: estimates $\hat{\rho}_{t,v}(v_i)$ of heat kernel pagerank values for vertices v_i hosted in the network

machine hosting seed vertex v generates $r = \frac{16}{\epsilon^3} \log n$ tokens t_i
$K \leftarrow c \cdot \frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}$ for any choice of $c \geq 1$
each token t_i does the following: pick a value k with probability $p_k = e^{-t}\frac{t^k}{k!}$, then hold the counter value $k_i \leftarrow \min\{k, K\}$
for iterations $j = 1 \ldots K$ **do**
    every machine performs the following in parallel:
    for every token t_i the machine currently holds **do**
        if $k_i == j$ **then**
            hold on to this token for the duration of the iterations
        else
            send t_i to a random neighbor as a message over a communication link
        end if
    end for
end for
let C_i be the number of tokens machine i currently holds
each machine with $C_i > 0$ returns C_i/r as an estimate for $\rho_{t,v}(v_i)$ of the vertex v_i it hosts

By Theorem 1, an ϵ-approximate heat kernel pagerank can be computed with the above procedure by setting $r = \frac{16}{\epsilon^3} \log n$ in the centralized setting. Further, the approximation guarantee holds when limiting the maximum length of random walks to $K = O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$, so that each token is passed for $\max\{k, K\}$ rounds, where k is drawn with probability p_k as described above. In the centralized setting,

this limit keeps the running time down.

In contrast, the distributed algorithm `DistributedApproxHKPRseed` takes advantage of decentralized control to take multiple random walk steps via multiple edges at a time. That is, through parallel execution, the running time depends only on the length of random walks, whereas when running the random walks in serial the running time must also include the number of random walks performed. Thus, keeping K small is critical in keeping the number of rounds low, and is the key to the efficiency of our local clustering algorithms. The correctness of the algorithm follows directly from Theorem 1.

Theorem 17. *For any graph G hosted by a network in the CONGEST model, any seed vertex $v \in V$, and any error bound $0 < \epsilon < 1$, the distributed algorithm* ***`DistributedApproxHKPRseed`*** *outputs an ϵ-approximate heat kernel pagerank with probability at least $1 - \epsilon$.*

The correctness of the algorithm holds for any choice of t, and in fact we use a particular value of t in our local clustering algorithm (see Section 6.4). Regardless, it is clear that the running time is independent of any choice of t. In fact, we demonstrate in the proof of Theorem 18 that it is independent of n as well.

Theorem 18. *For any graph G hosted by a network in the CONGEST model, any seed vertex $s \in V$, and any error bound $0 < \epsilon < 1$, the distributed algorithm* ***`DistributedApproxHKPRseed`*** *finishes in $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$ rounds.*

Proof. We show that there is no congestion in the network during any round of the algorithm; i.e., there are never more than $O(\log n)$ bits sent over any communication link in any iteration of the random walk process. The proof then follows since each step of the random walk requires only one round of communication.

In any run of the algorithm, $\frac{16}{\epsilon^3}\log n$ tokens are created, each holding a message k_i corresponding to a random walk length. The token contains no other information. In particular, no machine or vertex IDs are transmitted through the tokens. Therefore passing a token involves sending a message of constant size in any iteration of the algorithm. In the worst case, every token is transmitted through a single link in a single iteration of the algorithm. However, this is still only $O(\frac{16}{\epsilon^3}\log n)$ bits, and so meets the constraints of the model. Namely, even the worst case of sending every token over one link can be done with a single round of communication. Therefore any random walk step requires only one round of communication, and by construction at most $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$ random walk steps are performed in the algorithm. □

6.4 Distributed Local Cluster Detection

In this section we present a fast, distributed algorithm for the local cluster detection problem. The algorithm is a distributed version of the sweep algorithm used for local cluster detection in the centralized setting, which involves investigating sets of vertices which accumulate in decreasing order of their $\hat{\rho}_{t,v}(v_i)/d_i$ values. The process is efficient and requires at most one linear scan of the machines in the network (we actually show that the process can be much faster).

As in the centralized setting, the sweep requires an ordering of the vertices in decreasing order of $\hat{\rho}_{t,v}(v_i)/d_i$. Then the majority of the work of the algorithm is investigating sufficiently many of the $n-1$ cuts (S_j, S_j^C) given by the first j vertices in the ordering and the last $n-j$ vertices in the ordering, respectively, for $j = 1, \ldots, n-1$. However, by "sufficiently many" we indicate that we may stop investigating the cut sets when either the volume or the size of the segment S_j is

large. Assume this point is after $j = \underline{j}$. Then we choose the cut set that yields the minimum Cheeger ratio among the $\underline{j}$ possible cut sets.

Algorithm 12. `DistributedLocalCluster`$(G, v, \sigma, \varsigma, \phi, \epsilon)$

input: a graph G hosted by a network in the CONGEST model, a seed vertex v, a target cluster size σ, a target cluster volume ς, an optimal Cheeger ratio ϕ, an error bound ϵ

output: a set of vertices S with $\Phi(S) \in O(\sqrt{\phi})$

1: $t \leftarrow \phi^{-1} \log(\frac{2\sqrt{\varsigma}}{1-\epsilon} + 2\epsilon\sigma)$
2: $\hat{\rho} \leftarrow$ `DistributedApproxHKPRseed`(G, v, t, ϵ)
3: every machine hosting a vertex v_i with a non-zero value for $\hat{\rho}[v_i]$ sends $\hat{\rho}[v_i]/d_i$ to every other machine hosting a vertex in the support of $\hat{\rho}$ ▷ ***Phase 1***
4: sort the vertices in the support of $\hat{\rho}$ according to $\hat{\rho}[v_i]/d_i$ ▷ ***Phase 1***
5: compute Cheeger ratios of each of the cut sets with a call of the **Distributed Sweep Algorithm** ▷ ***Phase 2***
6: output the cut set of minimum Cheeger ratio ▷ ***Phase 2***

In the centralized setting, this process will take $O(n \log n)$ time in general. The authors in [29] give a distributed sweep algorithm that finishes in $O(n)$ rounds. We improve the analysis of [29] using heat kernel pagerank. The running time of our sweep algorithm is given in Lemma 9.

The sweep involves two phases. Let π be the ordering of vertices in decreasing order of $\hat{\rho}[v_i]/d_i$. In Phase 1, the goal is for each machine to know the place of its vertex in π. Each machine can compute the $\hat{\rho}_{t,v}(v_i)/d_i$ value for its hosted vertex v_i locally, and we use $O(\frac{1}{\epsilon})$ rounds to ensure each machine knows the π values of all other vertices (see the proof of Lemma 9). In Phase 2, we use the decentralized sweep of [29] described presently:

Distributed Sweep Algorithm. Let $N := N_{\hat{\rho}} = |\text{supp}(\hat{\rho}_{t,v})|$ denote the number of vertices with a non-zero estimated heat kernel pagerank value after running the algorithm `DistributedApproxHKPRseed`. Assume each machine knows the ordering π after Phase 1. We will refer to vertices by their place in the ordering.

Define S_j to be the cut set of the first j vertices in the ordering. Then computing the Cheeger ratio of each cut set S_j involves a computation of $\text{vol}(S_j)$ as well as $|\partial(S_j)|$. Define the following:

- L_j^π is the number of neighbors of vertex v_j in S_{j-1}, and
- R_j^π is the number of neighbors of vertex v_j in S_j^C.

Then the Cheeger ratio of each cut set can be computed locally by:

$$\circ\ |\partial(S_j)| = |\partial(S_{j-1})| - L_j^\pi + R_j^\pi, \text{ with } |\partial(S_1)| = d_1 \tag{6.1}$$

$$\circ\ \text{vol}(S_j) = \text{vol}(S_{j-1}) + L_j^\pi + R_j^\pi, \text{ with } vol(S_1) = d_1. \tag{6.2}$$

We now show that a sweep can be performed in $O(N)$ rounds. Each machine knows the IDS of its neighbors and thereby knows the degree of its hosted vertex and the identities of neighboring vertices. After Phase 1 each machine knows the place of every vertex in the ordering π. Therefore, each machine can compute locally if a neighbor is in S_{j-1} or S_j^C, and so L_j^π and R_j^π can be computed locally for each vertex v_j. Each machine can then prepare an $O(\log n)$-bit message of the form $(\text{ID}, L_j^\pi, R_j^\pi)$. Each of the N messages of this form can then be sent to one machine using the upcasting algorithm (described in the proof of Lemma 9) using the π ordering as node rank. We note that the N machines in the ordering are necessarily in a connected component of the network, and so the upcasting procedure can be performed in $O(N)$ rounds. Finally, once the first machine in the ordering is in possession of the ordering π, and the values of (L_j^π, R_j^π) for every vertex, it may iteratively compute $\Phi(S_j)$ locally using the rules (6.1) and (6.2). Thus, this machine can output the minimum Cheeger ratio ϕ^* as well as the j^* such that $\Phi(S_{j^*}) = \phi^*$ after $O(N)$ rounds.

Lemma 9. *Performing Phases 1 and 2 of a distributed sweep takes $O(\frac{1}{\epsilon})$ rounds.*

Proof. First we describe how to send N $O(\log n)$-sized messages to a single machine in $O(N)$ rounds of communication. For this we can use the upcasting algorithm of [80] (as described in [29]). We first construct a priority BFS tree of the N machines hosting vertices in $\text{supp}(\hat{\rho})$. We emphasize again that these machines are necessarily in a connected component of the network, and it is shown in [80] that such a BFS tree can be constructed in $O(N)$ time. Each node (machine) in the tree then upcasts its message to the root node through the edges of the tree.

In Phase 1, the vertices need to be sorted according to their (non-zero) π values. In this case, the machines use the value of $\hat{\rho}_{t,v}(v_i)/d_i$ for their hosted vertices v_i as their tree node rank so that the machine with the highest $\hat{\rho}_{t,v}(v_i)/d_i$ value is the root of the tree. Then each node upcasts its $\hat{\rho}_{t,v}(v)/d_v$ value to the root through the edges of the tree. The root node locally sorts these values and then floods all the π values to the nodes through tree edges. The upcast and flooding process take $O(N)$ rounds to reach each of the nodes in the tree.

Phase 2 consists of the **Distributed Sweep Algorithm**, where the first machine in the ordering computes the Cheeger ratio of each segment S_j. In order to send each of the $(\text{ID}, L_j^\pi, R_j^\pi)$ messages to the first machine of the ordering we again upcast through the edges of a priority BFS tree, and in this round we use π values as node rank. The root node is then able to locally compute Cheeger ratios and output the cutset of minimum Cheeger ratio after $O(N)$ rounds for upcasting.

Thus Phase 1 requires $O(N)$ rounds for upcasting and flooding values. Phase 2 requires $O(N)$ rounds for upcasting values necessary for locally computing Cheeger ratios. Since we compute an ϵ-approximate heat kernel pagerank vector as our distribution, we know that N is no more than $O(\frac{1}{\epsilon})$. This is because we assume $\sum_{v \in V} \rho_{t,v}(v) = 1$, and so no more than $\frac{1}{\epsilon}$ vertices can have values at least ϵ. Thus

the full sweep takes $O(\frac{1}{\epsilon})$ rounds. □

We note here that the time required for the sweep may be reduced if there are size or volume restraints for the local cluster. In this case, an alternative distributed sweep algorithm may be utilized. As usual, we refer to each machine by the place of their hosted vertex in the ordering π. Machine 1 begins the sweep by sending $\text{vol}(S_1), |\partial(S_1)|$ to machine 2. Then machines $j = 2, \ldots, N$ iteratively compute $\Phi(S_j)$ using the values of $\text{vol}(S_{j-1}), |E(S_{j-1}, S_{j-1}^C)|, L_j^\pi$ and R_j^π, and then subsequently sending $\text{vol}(S_j), |\partial(S_j)|$ to the next machine $j+1$ in the ordering. Additionally, each machine can send the minimum Cheeger ratio ϕ^* computed thus far as well as the j^* such that $\Phi(S_{j^*}) = \phi^*$. Thus the last machine in the ordering can output S_{j^*}. In this algorithm, each iteration j will require d rounds of communication, were d is the shortest path distance between nodes $j-1$ and j. However, no two machines will ever be at a distance of greater than $O(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})})$ steps by construction. In this way, the first j Cheeger ratios can be computed in $O(j\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})})$ rounds.

If size or volume restraints are placed on the cluster, we may stop the sweep at machine $\underline{\text{j}}$ when the size or volume of $S_{\underline{\text{j}}}$ is greater than some specified value. We output the set S_{j^*} for that iteration, and this process requires $O(\underline{\text{j}}\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})})$ rounds.

The algorithm `DistributedLocalCluster` (Algorithm 12) is a complete description of our distributed local clustering algorithm. The correctness of the algorithm follows directly from Theorem 4.

Theorem 19. *For any graph G hosted by a network in the CONGEST model, suppose there is a set S with $vol(S) \leq vol(G)/4$, $|S| = \sigma$, and Cheeger ratio $\Phi(S) \leq \phi$. Then at least half of the vertices in S can serve as the seed v so that for*

any error bound $0 < \epsilon < 1$*, the algorithm* ***DistributedLocalCluster*** *will find a set of Cheeger ratio* $O(\sqrt{\phi})$ *with probability at least* $1 - \epsilon$*.*

Theorem 20. *For any graph* G *hosted by a network in the CONGEST model, any seed vertex* $v \in V$*, and any error bound* $0 < \epsilon < 1$*,* ***DistributedLocalCluster*** *finishes in* $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})} + \frac{1}{\epsilon}\right)$ *rounds.*

Proof. The only distributed computations are those for computing approximate heat kernel pagerank values (line 2) and Phase 1 (lines 3 and 4) and Phase 2 (line 5) of the distributed sweep. Computing heat kernel pagereank values takes $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$ rounds by Theorem 18, and Phases 1 and 2 together take $O(\frac{1}{\epsilon})$ rounds by Lemma 9. Thus the running time follows. □

One possible concern with the algorithm `DistributedLocalCluster` is that one cannot guarantee knowing the value of ϕ ahead of time. Therefore a true local clustering algorithm should be able to proceed without this information. This can be achieved by "testing" a few values of ϕ (and fixing some reasonable values for σ and ς). Namely, begin with $\phi = 1/2$ and run the algorithm above. If the output cut set S satisfies $\Phi(S) \in O(\sqrt{\phi})$, we are done. If not, halve the value of ϕ and continue. There are $O(\log n)$ such guesses, and we have arrived at the following.

Theorem 21. *For any graph* G *hosted by a network in the CONGEST model, any vertex* v*, and any error bound* $0 < \epsilon < 1$*, there is a distributed algorithm that computes a set* S *with Cheeger ratio within a quadratic of the optimal which finishes in* $O\left(\frac{\log(\epsilon^{-1})\log n}{\log\log(\epsilon^{-1})} + \frac{1}{\epsilon}\log n\right)$ *rounds.*

In particular, when ignoring polylogarithmic factors, the running time is $\tilde{O}\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})} + \frac{1}{\epsilon}\right)$.

6.5 Computing Local Clusters in the k-Machine Model

In this section we consider a graph on n vertices which is distributed across k machines in a computer network. This is the k-machine model introduced in Section 6.2.

In the k-machine model, we consider a network of $k > 1$ distinct machines that are pairwise interconnected by bidirectional point-to-point communication links. Each machine executes an instance of a distributed algorithm. The computation advances in rounds where, in each round, machines can exchange messages through their communication links. We again assume that each link has a bandwidth of $O(\log n)$ meaning that $O(\log n)$ bits may be transmitted through a link in any round. We also assume no shared memory and no other means of communication between machines. When we say an algorithm solves a problem in x rounds, we mean that x is the maximum number of rounds until termination of the algorithm, over all n-vertex, m-edge graphs G.

In this model we are solving massive graph problems in which the vertices of the graph are distributed among the k machines. We assume $n \geq k$ (typically $n \gg k$). Initially the entire graph is not known by a single machine but rather partitioned among the k machines in a "balanced" fashion so that the vertices and/or edges are partitioned approximately evenly among the machines. There are several ways of partitioning vertices, and we will consider a random partition, where vertices and incident edges are randomly assigned to machines. Formally, each vertex v of G is assigned independently and randomly to one of the k machines, which we call the host machine of v. The host machine of v thereafter knows the ID of v as well as the IDs and host machines of neighbors of v.

In the remainder of this section we prove the existence of efficient algorithms for computing heat kernel pagerank and computing local clusters in the k-machine model. Our main tool is the Conversion Theorem of [47].

Define M as the *message complexity*, the worst case number of messages sent in total during a run of the algorithm. Also define C as the *communication degree complexity*, or the maximum number of messages sent or received by any machine in any round of the algorithm. Then we use as a key tool the Conversion Theorem as restated below.

Theorem 22 (Conversion Theorem [47])**.** *Suppose there is an algorithm A_C that solves problem P in the CONGEST model for any n-vertex graph G with probability at least $1-\epsilon$ in time $T_C(n)$. Further, let A_C use message complexity M and communication degree complexity C. Then there exists an algorithm A_k that solves P for any n-vertex graph G with probability at least $1-\epsilon$ in the k-machine model in $\tilde{O}\left(\frac{M}{k^2}+\frac{T_C(n)C}{k}\right)$ rounds with high probability.*

In the forthcoming theorems, by "high probability" we mean with probability at least $1-1/n$.

We note that the proof of the Conversion Theorem is constructive, describing precisely how an algorithm A_k in the k-machine model simulates the algorithm A_C in the CONGEST model. We omit the simulation here but encourage the reader to refer to the proof for implementation details.

By Theorem 18, we know that heat kernel pagerank values can be estimated with ϵ-accuracy in $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$ rounds. A total of $O\left(\frac{16}{\epsilon^3}\log n\right)$ messages are generated and propagated for at most $O\left(\frac{\log(\epsilon^{-1})}{\log\log(\epsilon^{-1})}\right)$ random walk steps, for a total of $O\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3\log\log(\epsilon^{-1})}\right)$ messages sent during a run of the algorithm. In the first random walk step, each of the $O\left(\frac{16}{\epsilon^3}\log n\right)$ messages may be passed to a neighbor of the

seed vertex, so the message complexity is $O\left(\frac{16}{\epsilon^3}\log n\right)$. Therefore we arrive at the following.

Theorem 23. *There exists an algorithm that computes an ϵ-approximate heat kernel pagerank for any n-vertex graph in the k-machine model with probability at least $1-\epsilon$ and runs in $\tilde{O}\left(\frac{\log(\epsilon^{-1})}{\epsilon^3 k\log\log(\epsilon^{-1})}(\frac{1}{k}+1)\right)$ rounds with high probability.*

By Theorem 21, a local cluster about any seed vertex can be computed in $O\left(\frac{\log(\epsilon^{-1})\log n}{\log\log(\epsilon^{-1})}+\frac{1}{\epsilon}\log n\right)$ rounds. The message complexity for the heat kernel pagerank phase is $O\left(\left(\frac{\log(\epsilon^{-1})\log n}{\epsilon^3\log\log(\epsilon^{-1})}\right)\log n\right)$ and for the sweep phase is $O\left(\frac{1}{\epsilon}\log n\right)$, for a total message complexity of $O\left(\frac{\log(\epsilon^{-1})\log^2 n}{\epsilon^3\log\log(\epsilon^{-1})}+\frac{1}{\epsilon}\log n\right)$. The communication degree complexity is $O\left(\frac{16}{\epsilon^3}\log n\right)$ for the heat kernel pagerank phase (as above), and $O(\Gamma)$, where Γ is the maximum degree in the graph, for the sweep phase. Thus the communication degree complexity for the algorithm is the maximum of these two. We therefore have the following result for the k-machine model.

Theorem 24. *There exists an algorithm that computes a local cluster for any n-vertex graph in the k-machine model with probability at least $1-\epsilon$ and runs in $\tilde{O}\left(\frac{\log(\epsilon^{-1})}{\epsilon^3k^2\log\log(\epsilon^{-1})}+\frac{1}{\epsilon k^2}+\left(\frac{\log(\epsilon^{-1})}{k\log\log(\epsilon^{-1})}+\frac{1}{k\epsilon}\right)\max\left\{\frac{1}{\epsilon^3},\Gamma\right\}\right)$ rounds, where Γ is the maximum degree in the graph, with high probability.*

Acknowledgements

Chapter 6, in part, is a reprint of the material as it appears in the proceedings of *Algorithms and Models for the Web Graph*. Fan Chung and Olivia Simpson, LNCS 9479, 2015 pp. 177-189. The dissertation author was a primary investigator and author of this paper.

Chapter 7

Ranking on Evolving Graphs

Graphs are a powerful way of storing data with relational properties. By studying the graph we are able to reason about quality of relationships and underlying structure of the data. Over the course of this dissertation we have also seen how to unearth local structure in data by examining local areas of the graph. However, in studying the data in this way we are ignoring a potentially valuable quality of data – how it changes over time.

In some respects, graph analysis is an examination of only a most recent snapshot of the data. There are many cases, however, in which including a more complete history of the data can illuminate qualities that may otherwise be hidden. As an example, consider the Wikipedia graph in which links between articles are represented by directed edges. This graph is far from fixed and its evolution tends to correspond with major events in politics or pop culture. In 2011, for instance, comedian and television host Stephen Colbert called upon his fans to edit the Wikipedia article for "Bell" to reflect an erroneous recount of Paul Revere's famous ride as told by former Governor of Alaska, Sarah Palin [1]. After Wikipedia caught wind of the calculated edits, they locked the page and reverted it to its state before Colbert's call. This rapid and arguably notable evolution in the graph would go unseen without including history.

In this chapter, we present an algorithm for detecting local changes that occur over time. We measure changes in the graph as changes in rank with respect to a fixed "seed" vertex. Our method uses random walks on evolving graphs, and thus is adaptable to graphs which are large, heterogeneous, or noisy. Our method also allows for varying definitions of passage of time. That is, the amount of time for which a change should sustain is a parameter of the algorithm and allows for a mechanism to control complexity. Specifically, we show that for evolving graphs on n vertices with a history of length T, sampling $O(\frac{T}{\tau\delta}\log n)$ random walks is enough to detect a change of size δ which sustains for a span of time τ with high probability. Our methods can be used for detecting changes on scales both large and small, and have potential for applications in the space of anomaly detection, fraud detection, and cybersecurity among others.

7.1 Previous Work on Evolving Graphs

The idea of evolving or changing graphs is not new and has been explored in a number of contexts. Nicosia et al. [72] extend classical graph theoretic definitions and graph metrics to graphs that evolve over time and, in particular, define connectedness and components in an evolving graph they refer to as time-varying. In [4], the authors define a framework for an evolving graph in which in any particular time step a random edge is removed and a random pair of vertices are connected by an edge. In this paper they present algorithms for path connectivity and minimum spanning trees for graphs in this framework. The authors of [90] precisely define and analyze random walks with various policies on networks which change over time, and similarly Avin et al. [8] investigate the cover time of random walks on a graph in which an adversary is adding and removing edges over time. Finally, Bahmani et al. [11] present a way to compute and update PageRank on an

evolving graph.

7.2 Evolving Graphs

An evolving graph models data with both relational and temporal dynamics. Specifically, edges will have clocks attached to them indicating their time of birth and their time of death. We may alternatively allow for vertices to have separate temporal dynamics if isolated vertices are allowed. For the purposes of this chapter, we fix the vertex set and observe evolution on the edge set.

The evolution on the edge set is described by a list of contacts between vertices along with the time of initial contact and the duration of the contact. That is, edges are now defined as a four-tuple (v, v', t_b, t_d) which indicates that vertices v and v' were in contact from times t_b to t_d. These times can be viewed as the birth and death times of the edges, respectively. This brings us to our definition of a continuous time evolving graph.

Definition 8. Fix a vertex set V and some history of observation $[0, T]$. Then a *continuous time evolving graph* is a list of contacts among the vertices where each contact is of the form (v, v', t_b, t_d) for $[t_b, t_d] \subseteq [0, T]$ and represents an edge (v, v') which exists exactly from time t_b to t_d.

We can characterize an evolving graph by probing it at a specific moment of time, t_i. That is, for a time t_i, the graph G_{t_i} is the graph on the fixed vertex set and all edges (v, v', t_b, t_d) such that $t_b \leq t_i < t_d$. Such a graph G_{t_i} is an aggregation of contacts that existed at time t_i, and a sequence of such graphs is a discrete notion of the evolving graph given in Definition 8. With this, we give a precise definition of a discrete time evolving graph.

Definition 9. Fix a vertex set V. Then a *discrete time evolving graph* is a sequence of graphs $\mathfrak{G} = \{G_{t_i}\}_{i=0}^{K}$ at discrete times $\{t_i \in [0,T]\}_{i=0}^{K}$.

We allow multiple contacts between a pair of vertices, so that we may have edges $(v, v', t_b, t_d), (v, v', t'_b, t'_d)$ present in the list of contacts (so long as the intervals $[t_b, t_d], [t'_b, t'_d]$ do not overlap).

An important problem is the best way of discretizing an evolving graph. For instance, as stated in Definition 9, we may identify a graph in the sequence to a single time point t_i. However, it may be more meaningful to identify a time window $t_i := [t_i, t_i + \epsilon]$ and aggregate contacts which are active within that time window as a graph in the sequence. We assume this strategy has already been applied and that the discretization of the evolving graph reflects this.

7.3 Dynamic Ranking With Random Walks

The goal of this chapter is to develop an algorithm for detecting large changes on evolving graphs. For a fixed seed vertex v and a point of time t_i, we let $\rho^i_{t,v}(v_k)$ be the heat kernel pagerank value for vertex v_k at time t_i. We will use these values to "score" the vertices. Then, if $|\rho^i_{t,v}(v_k) - \rho^j_{t,s}(v_k)| > \delta$ for some specified $\delta > 0$, we define a large change for vertex v_k from time t_i to time t_j.

We will simplify notation for the purposes of this chapter and let $\rho_i(v)$ be the score of vertex v relative to a fixed seed vertex v' at time t_i, so that $\rho_i(v) := \rho^i_{t,v'}(v)$. We will always assume a fixed seed vertex and a fixed parameter t for the heat kernel pagerank computations $\rho_{t,v'}$ so that we refer to the scores only by the graph G_i and the vertex, v, being scored.

Now we precisely define the problem. Consider some lapse of time T. Our goal is to detect a local evolution by detecting drastic changes of score that persist

over some time window, $\tau := [t_0, t'] \subseteq [0, T]$. That is, we want to detect the situation:

$$|\rho_i(v) - \rho_j(v)| > \delta > 0,\ t_j - t_i \geq \tau, \tag{7.1}$$

in which the score of vertex v changes by at least δ in over the course of τ time.

Let us return to the example of the Wikipedia article graph. Consider the ranking of Wikipedia articles with respect to the fixed article for Brad Pitt. In the time preceding the year 2005, we would expect Brad Pitt's then-wife Jennifer Aniston to have a high score, and thereby high rank. However, around and after 2005, when Brad Pitt and Angelina Jolie began dating, we will observe that Jolie's rank grows while Aniston's rank drops. If we center the time window τ around the year 2005, we should be able to see that for some $j > i$, $\rho_i(aniston) > \rho_j(aniston)$ while $\rho_i(jolie) < \rho_j(jolie)$.

Using random walk-based algorithms for scoring is a natural choice as they model the probability of a random walker reaching a particular vertex. Here we assume that the time an average readers spends on Wikipedia is much smaller than the average time for a change in the Wikipedia graph to occur, and that the choice in evolving graph discretization reflects this. In the above scenario involving Brad Pitt, Jennifer Aniston, and Angelina Jolie, we model the probability that a Wikipedia user looking at Brad Pitt's article and subsequently clicking through links on Wikipedia at a particular moment in time lands on Aniston's article or Jolie's article. Noticeable changes in the likelihood of landing at Aniston's article over time might indicate significant changes in link structure corresponding to fewer connections between the newly-divorced couple, or even more connections between the newly-wed couple.

7.4 Detecting Big Changes on Evolving Graphs

In this section we prove the following statement showing that a sublinear number of random walkers in the size of the vertex set are enough to capture situation (7.1) with high probability, should it exist in the data. Note that the required number of random walkers $O(\frac{T}{\tau\delta}\log n)$ also depends on the parameters δ, the magnitude of change, and τ the period of time over which the change persists.

Lemma 10. *For an evolving graph $\mathfrak{G}$ and for a fixed seed vertex v', assume that for some times t_i, t_j with $t_j - t_i \geq \tau > 0$ the heat kernel pagerank score of a vertex v changes by $|\rho_i(v) - \rho_j(v)| > \delta > 0$. Then this change in score can be detected by sampling $R = O(\frac{T}{\tau\delta}\log n)$ random walks in $\mathfrak{G}$ with probability at least $1 - o(1)$.*

Proof. First, fix times t_i, t_j, the personalized starting distribution $s = \chi_{v'}$, the heat kernel pagerank parameter t, and the evolving vertex v. As we are using heat kernel pagerank to score vertices, we recall some ideas presented in the proof of Theorem 1 for computing heat kernel pagerank with random walks. Specifically, we consider launching a number of random walkers from the seed vertex v' where the length of the random walk k is drawn with probability $p_k = e^{-t}\frac{t^k}{k!}$. For a graph G_i, define the indicator random variable X_i to indicate that a random walk in G_i ends at vertex v, and similarly for X_j. Then the expectation of X_i is exactly $\mathbb{E}(X_i) = \rho_i(v)$ (and similarly $E(X_j) = \rho_j(v)$). Define $X = X_i - X_j$. Then we have that $\mathbb{E}(X) = \rho_i(v) - \rho_j(v) > \delta$. Now, let $\mathbb{X}$ be R_1 independent copies of X: $\mathbb{X} = \sum_{r=1}^{R_1} X$ and note that $\frac{1}{R_1}\mathbb{E}(\mathbb{X}) = \mathbb{E}(X)$. Set $R_1 = \frac{3}{\epsilon^2\delta}\log n$. Then, by the

multiplicative Chernoff bounds:

$$\begin{aligned}\Pr\left(\frac{1}{R_1}\mathbb{X} \geq (1+\epsilon)\mathbb{E}(X)\right) &= \Pr\left(\mathbb{X} \geq (1+\epsilon)\mathbb{E}(\mathbb{X})\right) \\ &\leq \exp\left(-(\epsilon^2/3)R_1\mathbb{E}(X)\right) \\ &\leq \exp\left(-(\epsilon^2\delta/3)R_1\right) \\ &\leq n^{-1}.\end{aligned}$$

Similarly, we have that

$$\begin{aligned}\Pr\left(\frac{1}{R_1}\mathbb{X} \leq (1-\epsilon)\mathbb{E}(X)\right) &\leq \exp\left(-(\epsilon^2/2)R_1\mathbb{E}(X)\right) \\ &\leq n^{-3/2}.\end{aligned}$$

Now we address how to sample times t_i, t_j. Assume we are drawing a time t_i uniformly at random from the interval $[0, T]$. Then the probability of sampling from a window of size τ is $\frac{\tau}{T}$. For R_2 independent trials, let $X_r = 1$ if the r^{th} sample is in the appropriate window and we count these by $X = \sum_{r=1}^{R_2} X_r$. Then the expected number of samples in the appropriate window is $R_2\frac{\tau}{T}$. In fact the expected number of samples in all windows of size τ is $R_2\frac{\tau}{T}$. Again, we can use the multiplicative Chernoff bounds to show that by choosing $R_2 \geq \frac{2T}{\epsilon^2\tau}\log(\epsilon^{-1})$,

$$\Pr\left(X \leq (1-\epsilon)\mathbb{E}(X)\right) = (1-\epsilon)\left(2\epsilon^{-2}\log(\epsilon^{-1})\right) \leq \epsilon.$$

We briefly address the problem of samples being "too clustered" about a single moment of time so as to not detect a change, even if we have enough samples in the particular time window. This corresponds to all of the expected number of samples lying in too small a radius, which we call $\tau_0 < \tau$. We show that this

probability is small, especially as we increase our error tolerance, ϵ.

$$\begin{aligned}
&\Pr(\forall(i,j),\ |t_j - t_i| \leq \tau_0) \\
&\quad = 1 - \Pr(|t_j - t_i| \geq \tau_0 \text{ for some } (i,j)) \\
&\quad = 1 - \binom{R_2 \frac{\tau}{T}}{2} \left(\frac{\tau - 2\tau_0}{\tau}\right) \\
&\quad \leq 1 - \left(1 - 2\frac{\tau_0}{\tau}\right) \left(2\epsilon^{-4}\log^2(\epsilon^{-1}) - \epsilon^{-2}\log(\epsilon^{-1})\right).
\end{aligned}$$

Then the total random walk samples (in terms of δ, T, τ, n) are $R = R_1 R_2 = O(\frac{T}{\tau\delta}\log n)$. □

This suggests an algorithm for detecting changes of size at least δ which persist for at least τ. Namely, we sample from the evolving graph $\mathfrak{G}$ for R_2 times, giving a collection of R_2 graphs G_{t_i}. Then, in each sampled graph we simulate R_1 random walkers from the seed vertex in order to compute heat kernel pagerank scores. We outline the algorithm DynamicRanking below.

As mentioned above, the key to Lemma 10 is the control allowed with the parameter δ, and especially the parameter τ, as we explain in the following illustrative examples.

Example. (Brangelina.) We first revisit the case of detecting the divorce of Brad Pitt and Jennifer Aniston and the marriage of Brad Pitt and Angelina Jolie. In this case, the particular event occurred in 2005 and persisted well beyond that[1]. Therefore, we may set our τ parameter to be rather large, resulting in the need for fewer samples. Then we simply need one sample before the year 2005 and one sample after 2005 to detect the change.

[1]We might anticipate further changes concerning the recent divorce of Brad Pitt and Angelina Jolie.

Algorithm 13. DynamicRanking($\mathfrak{G}, v', t, v, \delta, \tau, \epsilon$)

input: evolving graph $\mathfrak{G} = \{G_t\}_{t=0}^{T}$, seed vertex $v' \in V$, heat kernel pagerank parameter $t \in \mathbb{R}^+$, evolving vertex $v \in V$, magnitude of change $\delta > 0$, time of sustained change τ, error parameter $0 < \epsilon < 1$.

output: set of times (t_i, t_j) such that $|\rho_i(v) - \rho_j(v)| > \delta > 0,\ t_j - t_i \geq \tau$.

$R_1 \leftarrow \frac{3}{\epsilon^2\delta} \log n$
$R_2 \leftarrow \frac{2T}{\epsilon^2\tau} \log(\epsilon^{-1})$
for $l = 1$ to R_2 **do**
    sample time t_l from $[0, T]$ uniformly at random
    for R_1 iterations **do**
        Start
            simulate a P random walk from vertex v' where k steps are taken with probability $e^{-t}\frac{t^k}{k!}$
            let v be the last vertex visited in the walk
            $\rho_l(v) \leftarrow \rho(v) + 1/R_1$
        End
    end for
end for
initialize an empty list *times*
for all pairs i, j **do**
    if $|\rho_i(v) - \rho_j(v)| > \delta$ **then**
        add pair (t_i, t_j) to list *times*
    end if
end for
return *times*

Example. (Colbert.) As a reminder, in 2011, Stephen Colbert called upon his fans to edit the Wikipedia "Bell" article to reflect an erroneous recount of Paul Revere's famous ride told by Sarah Palin [1]. After Wikipedia caught wind of the calculated edits, they locked the page and reverted it to its state before Colbert's call. In this case we are concerned with detecting rapid changes in a short period of time. Therefore, we would set τ to be small, resulting in more samples and more, smaller windows of time. Then we would see the changes in the window surrounding the event that are distinct from "normal" evolutionary changes in other windows.

Finally we note that for $\delta > \epsilon$ we can use the guarantees of the heat kernel pagerank approximation algorithm `ApproxHKPRseed` and replace the random walk procedure above with the line:

$$\rho_l(v) \leftarrow \texttt{ApproxHKPRseed}(G_l, t, v', \epsilon)(v)$$

and we have the following corollary at our disposal.

Corollary 4. *For an evolving graph $\mathfrak{G}$ and for a fixed seed vertex v', assume that for some times t_i, t_j with $t_j - t_i \geq \tau > 0$ the heat kernel pagerank score of a vertex v changes by $|\rho_i(v) - \rho_j(v)| > \delta > 0$. Then this change in score can be detected using heat kernel pagerank random walks with no more than $O\left(\log(\epsilon^{-1})/\log\log(\epsilon^{-1})\right)$ steps for each random walker. Using personalized PageRank requires no more than $O\left(\log_{\frac{1}{(1-\alpha)}}(4/\epsilon)\right)$ steps per random walker, where α is the restart constant.*

The result on heat kernel pagerank follows from the results of Chapter 2 and the result for personalized PageRank is due to [17].

Acknowledgements

Chapter 7, is currently being prepared for submission for publication of the material. Olivia Simpson, Christine Klymko. The dissertation author was the primary investigator and author of this material.

Bibliography

[1] Colbert defends sarah palin's paul revere story by having fans edit wikipedia's bell entry. http://www.huffingtonpost.com/2011/06/07/colbert-sarah-palin-paul-revere-video_n_872375.html.

[2] Rudolf Ahlswede and Andreas Winter. Strong converse for identification via quantum channels. *IEEE Transactions on Information Theory*, 48(3):569–579, 2002.

[3] Noga Alon and Vitali D. Milman. λ_1, isoperimetric inequalities for graphs, and superconductors. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[4] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Proceedings of the 3rd ACM Innovations in Theoretical Computer Science Conference*, pages 149–160. ACM, 2012.

[5] Reid Andersen and Fan Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In *Proceedings of Theory and Applications of Models of Computation*, pages 1–12. Springer, 2007.

[6] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486. IEEE, 2006.

[7] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 235–244. ACM, 2009.

[8] Chen Avin, Michal Kouckỳ, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *International Colloquium on Automata, Languages, and Programming*, pages 121–132. Springer, 2008.

[9] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45(2):890–904, 2007.

[10] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[11] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 24–32. ACM, 2012.

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

[13] Michele Benzi and Christine Klymko. Total communicability as a centrality measure. *Journal of Complex Networks*, 1(2):124–149, 2013.

[14] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, and Richard Peng. Near linear-work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. In *Proceedings of the 23th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 13–22. ACM, 2011.

[15] Erik Boman and Bruce Hendrickson. On spanning tree preconditioners. *Manuscript, Sandia National Laboratories*, 3, 2001.

[16] Erik Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3):694–717, 2003.

[17] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Shang-Hua Teng. A sublinear time algorithm for pagerank computations. In *International Workshop on Algorithms and Models for the Web Graph*, pages 41–53. Springer, 2012.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998.

[19] Pak K. Chan, Martine D.F. Schlag, and Jason Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096, 1994.

[20] Long Cheng, Hanlei Wang, Zeng-Guang Hou, and Min Tan. Reaching a consensus in networks of high-order integral agents under switching directed topologies. *International Journal of Systems Science*, 47(8):1966–1981, 2016.

[21] Fan Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[22] Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

[23] Fan Chung. A local graph partitioning algorithm using heat kernel pagerank. *Internet Mathematics*, 6(3):315–330, 2009.

[24] Fan Chung and Mary Radcliffe. On the spectra of general random graphs. *The Electronic Journal of Combinatorics*, 18(P215):1, 2011.

[25] Fan Chung and Olivia Simpson. Solving linear systems with boundary conditions using heat kernel pagerank. In *International Workshop on Algorithms and Models for the Web Graph*, pages 203 – 219. Springer, 2013.

[26] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log 1/2 n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 343–352. ACM, 2014.

[27] Jorge Cortés and Francesco Bullo. Coordination and geometric optimization via distributed dynamical systems. *SIAM Journal on Control and Optimization*, 44(5):1543–1574, 2005.

[28] Demetres Cristofides and Klas Markström. Expansion properties of random cayley graphs and vertex transitive graphs via matrix martingales. *Random Structures Algorithms*, 32(8):88–100, 2008.

[29] Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Distributed computation of sparse cuts via random walks. In *Proceedings of the International Conference on Distributed Computing and Networking*, pages 6:1–6:10, 2015.

[30] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed pagerank computation. In *Proceedings of the International Conference on Distributed Computing and Networking*, pages 11–26. Springer, 2013.

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

[32] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[33] William E. Donath and Alan J. Hoffman. Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15(3):938–944, 1972.

[34] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.

[35] J. Alexander Fax and Richard M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9):1465–1476, 2004.

[36] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

[37] George E. Forsythe and Richard A. Leibler. Matrix inversion by a monte carlo method. *Mathematical Tables and Other Aids to Computation*, 4(31):127–129, 1950.

[38] Shayan Oveis Gharan and Luca Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *Proceedings of 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 187–196. IEEE, 2012.

[39] Gene H. Golub and Michael L. Overton. The convergence of inexact chebyshev and richardson iterative methods for solving linear systems. *Numerische Mathematik*, 53(5):571–593, 1988.

[40] Keith D. Gremban, Gary L. Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 65–69. IEEE, 1995.

[41] David Gross. Recovering low-rank matrices from few coefficients in any basis. *IEEE Transaction on Information Theory*, 57:1548–1566, 2011.

[42] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, synamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[43] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[44] Anil Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[45] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[46] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time.

In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 911–920. ACM, 2013.

[47] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 391–410. SIAM, 2015.

[48] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *CEAS*, 2004.

[49] Kyle Kloster and David F. Gleich. A nearly-sublinear method for approximating a column of the matrix exponential for matrices from large, sparse networks. In *International Workshop on Algorithms and Models for the Web Graph*, pages 68–79. Springer, 2013.

[50] Ioannis Koutis and Gary L. Miller. A linear work $o(n^{1/6})$ time algorithm for solving planar laplacians. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1002–1011. ACM-SIAM, 2007.

[51] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *Proceedings of 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 235–244. IEEE, 2010.

[52] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for sdd linear systems. In *Proceedings of 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

[53] Valdis Krebs. New political patterns. http://www.orgnet.com/divided, 2008.

[54] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2013.

[55] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[56] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th ACM International Conference on World Wide Web*, pages 695–704. ACM, 2008.

[57] Jure Leskovec and Julian J. McAuley. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, pages 539–547, 2012.

[58] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. Isorankn: Spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12):i253–i258, 2009.

[59] Frank Lin and William W. Cohen. Power iteration clustering. In *Proceedings of the 27th International Conference on Machine Learning*, pages 655–662, 2010.

[60] Frank Lin and William W. Cohen. A very fast method for clustering big text datasets. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 303–308, 2010.

[61] László Lovász and Miklós Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 346–354. IEEE, 1990.

[62] László Lovász and Miklós Simonovits. Random walks in a convex body and an improved volume algorithm. *Random Structures & Algorithms*, 4(4):359–412, 1993.

[63] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, pages 340–349, 2010.

[64] David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobioloy*, 54:396–405, 2003.

[65] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[66] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 135–146. ACM, 2010.

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

[68] Kevin L. Moore, Tyrone Vincent, Fadel Lashhab, and Chang Liu. Dynamic consensus networks with application to the analysis of building thermal processes. In *Proceedings of 18th IFAC World Congress*, 2011.

[69] Mark Newman. Network data. http://www-personal.umich.edu/~mejn/netdata/, 2013.

[70] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002.

[71] Wei Ni and Daizhan Cheng. Leader-following consensus of multi-agent systems under fixed and switching topologies. *Systems & Control Letters*, 59(3–4):209–217, 2010.

[72] Vincenzo Nicosia, John Tang, Mirco Musolesi, Giovanni Russo, Cecilia Mascolo, and Vito Latora. Components in time-varying graphs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(2):023101, 2012.

[73] Reza Olfati-Saber and Richard M. Murray. Graph rigidity and distributed formation stabilization of multi-vehicle systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 3, pages 2965–2971. IEEE, 2002.

[74] Reza Olfati-Saber and Richard M. Murray. Consensus protocols for networks of dynamic agents. In *Proceedings of the 2003 American Control Conference*, pages 951–956, 2003.

[75] Reza Olfati-Saber and Richard M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004.

[76] Roberto Imbuzeiro Oliveira. Concentration of the adjacency matrix and of the laplacian in random graphs with independent edges. *arXiv preprint arXiv:0911.0600*, 2009.

[77] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{O}$(m)-time spectral algorithm for balanced separator. In *Proceedings of the 44th ACM Symposium on Theory of Computing*, pages 1141–1160. ACM, 2012.

[78] Fernando Paganini, John Doyle, and Steven Low. Scalable laws for stable network congestion control. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 1, pages 185–190 vol.1. IEEE, 2001.

[79] Gopal Pandurangan and Maleq Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook*, 2010.

[80] David Peleg. Distributed computing. *SIAM Monographs on Discrete Mathematics and Applications*, 5, 2000.

[81] Richard Peng and Daniel A. Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 333–342. ACM, 2014.

[82] Amirreza Rahmani, Meng Ji, Mehran Mesbahi, and Magnus Egerstedt. Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization*, 48(1):162–186, 2009.

[83] John H. Reif. Efficient approximate solution of sparse linear systems. *Computers & Mathematics with Applications*, 36(9):37–58, 1998.

[84] Sushant Sachdeva and Nisheeth K. Vishnoi. Matrix inversion is as easy as exponentiation. *arXiv preprint arXiv:1305.0526*, 2013.

[85] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[86] Daniel A. Spielman. Algorithms, graph theory, and linear equations in laplacian matrices. In *Proceedings of the International Congress of Mathematicians*, volume 4, pages 2698–2722, 2010.

[87] Daniel A. Spielman and Shang-Hua Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $o(m^{1.31})$. In *Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 416–427. IEEE, 2003.

[88] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of Computing*, pages 81–90. ACM, 2004.

[89] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008.

[90] Michele Starnini, Andrea Baronchelli, Alain Barrat, and Romualdo Pastor-Satorras. Random walks on temporal networks. *Physical Review E*, 85(5):056115, 2012.

[91] John Toner and Yuhai Tu. Flocks, herds, and schools: A quantitative theory of flocking. *Physical Review E*, 58:4828–4858, Oct 1998.

[92] Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012.

[93] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. UIUC manuscript, 1990.

[94] Nisheeth K. Vishnoi. Laplacian solvers and their algorithmic applications. *Theoretical Computer Science*, 8(1-2):1–141, 2012.

[95] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.