Fast Linear Solvers for Laplacian Systems

UCSD Research Exam

Olivia Simpson

Fall 2013

Solving a system of linear equations is a fundamental problem that has deep implications in the computational sciences, engineering, and applied mathematics. The problem has a long history and, until recently, has not broken polynomial time bounds. In this report we present a survey of the algorithms that solve symmetric diagonally dominant linear systems in near-linear time. We also discuss a new linear solver which uses random walk approximations to achieve sublinear time, assuming a random walk step takes unit time. The recent success of these algorithms has inspired their advocacy as a new class of algorithmic primitives. We discuss the progress being made to this end and present a few ways these fast linear solvers may be applied to various graph problems.

1 Introduction

Laplacian linear systems arise in a number of natural contexts including finding consensus in multi-agent networks [51], calculating a maximum flow [27], or characterizing the motion of coupled oscillators in a system [36]. In many of these cases, however, the graphs of the networks are so massive that traditional methods for solving the system are infeasible. The classic Gaussian elimination method, for example, takes time $O(n^3)$. Even the most recent improvements to $O(n^{2.3727})$ [60] are of limited benefit for graphs on millions of nodes and billions of edges, which are becoming increasingly common [44].

The problem of finding a solution to a system of equations is one of the oldest in the numerical and computational sciences. The most remarkable recent advances in the area solve linear systems in nearly-linear time – a huge jump from direct methods requiring polynomial time. The improvements are so monumental, in fact, that many researchers are advocating the use of these algorithms as primitives to be used in larger problems. The effort is gaining momentum and is referred to by some as the Laplacian Paradigm [58].

In this report we present a survey of the algorithms which broke complexity barriers as well as the theory behind them. The first algorithm to achieve nearly-linear time is the solver of Spielman and Teng [56], which uses a preconditioned iterative solver as a premise, but instead implements a recursive procedure. The best solvers succeeding the solver of Spielman and Teng were presented by Koutis, Miller, and Peng in [41, 42], and improve both the recursive procedure and the quality of preconditioner. Then we introduce a variation of the problem with boundary conditions, and present a new algorithm recently presented in [23] which adopts an entirely different method. Namely, the solver computes directly the matrix-vector product of the inverse of the coefficient matrix and the vector of constants by taking a sum of random walks on the graph. The algorithm does not involve computing the matrix inverse, only the matrix-vector product, and achieves sublinear time.

The remainder of the report is organized as follows. In Section 2, we provide some basic definitions and introduce the graph matrices we will be using. In Section 3 we give a more complete statement of the problem and outline the traditional methods for solving linear systems. Then we present the first nearly-linear solver of Spielman and Teng in Section 4, and the improvements of Koutis et al. in Section 5. In Section 6 we give a very recent linear solver which beats previous time bounds and uses a new method for computing the solution satisfying boundary conditions. Researchers involved with these projects are proponents of their application to various graph problems, and we discuss the potential for

application in Section 7. Finally, in Section 8, we address further research direction and give closing remarks. First, we conclude this section with an introduction to Laplacian systems and some historical results of spectral graph theory.

1.1 Why Laplacian systems?

When considering very large graphs, it is useful to have a notion of "closeness" for two given vertices without computing paths. One clever way to measure this is borrowed from the theory of electrical networks. Begin by modeling the graph as an electrical circuit. Edges correspond to wires and are weighted according to their *conductance* (the inverse of resistance), and voltages are set at the vertices. Two important properties will govern how voltages are set for an electrical flow. First, Kirchhoff's law states that the net flow at each vertex is zero, except for the points at which current is injected or extracted. Second, Ohm's law says that the current on an edge is proportional to the conductance of the wire times the difference in voltage of the two endpoints. Together, these properties tell us that the amount of current needed to maintain voltages across the network can be computed by solving a system of linear equations, $Ax = b$, where A is the coefficient matrix and b is the vector that is 1 in the component corresponding to the current injection point, -1 at the extraction point, and 0 elsewhere. The voltage values computed by solving the system then gives us a way of measuring vertex distance. Namely, if x is the vector of computed voltage values with a unit of current injected at vertex v_i and extracted at vertex v_j, then the distance from v_i to v_j can be computed by $|x[i] - x[j]|$. In the electrical network, this is known as the *effective resistance*.

Results of the last decade have shown that linear systems in the graph Laplacian can be efficiently solved with a spectral decomposition. While it may seem like a small class, it actually happens that Laplacian linear systems arise in a number of natural contexts. For example, in our computation of effective resistance, the coefficient matrix is indeed the Laplacian. In this case, we measure graph distance by solving a system in the Laplacian of the graph in question. Even more, it has been shown that any symmetric, diagonally dominant linear system can be reduced to a Laplacian system, so a large class of problems can be addressed with improvements to one solver.

Large, complex graphs arise naturally in a discrete world, from internet graphs and social networks to human genomes and flocking formations, and they exhibit a number of characteristics which give their representative matrices nice properties. Namely, these graphs tend to be sparse, prone to neat clusterings, and have vertex degrees which follow a power law distribution [22]. By developing a deeper understanding of the underlying matrices we become better equipped to study natural phenomena in a meaningful way.

1.2 A brief history of spectral graph theory

The practice of using matrix theory in graph analysis has a long history, with results dating as far back as the mid-nineteenth century. The matrix-tree theorem, for instance, states that the number of spanning trees in a graph is equal to any cofactor of the Laplacian, and was first proven by Kirchhoff in 1847 [40]. Chemistry also contributed early results with a spectral analysis to characterize molecular stability. In fact, chemists were among the first to formalize the ideas of graph theory with chemical graphs as early as 1867 [14].

The modern era demonstrates an inclination toward characterizing properties of graph spectra and the relationship with structure. The 1957 paper of Collatz and Sinogowitz [26] is a general survey of graph spectra. Early works of Fiedler [30, 31] outline a number of fundamental properties of the second smallest eigenvalue of the Laplacian matrix of a graph, consequently dubbed the *algebraic connectivity* or *Fiedler value*. Around the same time, Donath and Hoffman [28] show that certain cut ratios of a graph are bounded by functions of the Laplacian eigenvalues. An important consequence of this is a spectral bipartitioning heuristic. Bounds on Laplacian eigenvalues are also given by Cheeger in [20], and provide a foundational basis for defining good cuts in graphs. Later, Alon and Milman [6] and Mohar [45] develop the theory in studies of the relationship of this second eigenvalue and the isoperimetric number of the graph, which is deeply related to finding balanced cuts. In [3, 4] expansion properties of the graph are found to be related to the algebraic connectivity as well. Together these results are the basis of many spectral partitioning algorithms used today.

The advent of local spectral partitioning algorithms is a milestone in the history of spectral graph theory. Before, only heuristics were available. A local algorithm is one which only requires work to be on done on a portion of a graph, and runtimes are typically in terms of the size of this small portion. The local partitioning algorithm of [56] finds highly-connected sets by testing distributions of random walks. This idea is later improved by computing a vector associated to the Laplacian [9]. The quality of these cuts are a marked improvement, as they rely on mixing properties derived from vertex degrees.

As suggested, some of the biggest results of spectral graph theory concern the fundamental relationships between the spectra of graph matrices and properties such as the connectedness, bipartiteness, and bounds for partitions. The success and applicability of these techniques has since spawned a number of spectral algorithms for traditionally discrete graph problems. For example, a spectral decomposition is used in a graph coloring heuristic in [10]. Grimes et al. [35] give a first application of spectral properties to sparse matrix reordering problems in their algorithm for finding a pseudopheripheral node of a graph using the largest eigenvector of an adjacency matrix of the graph, and in [11] a spectral algorithm for computing an envelope reducing matrix reordering is presented. In [52], Pothen et al. adapt spectral edge separator techniques to find good vertex separators.

In 2007, Vladimir Nikiforov pioneered an effort to prove extremal properties of graphs using spectral methods. He began with a restatement of a result originally due to Nosal [50] that asserts that a graph with a large enough spectral radius must contain a triangle [46]. Nikiforov later extends this result to guarantee the existence of cycles of every length up to $n/320$ [47]. Nikiforov himself has focused on cliques and cycles [49, 15, 32], though the techniques employed may be used across a wide range of problems. In particular, a major result of [48] gives a way of identifying a well-connected neighborhood of a graph by removing a small number of low-degree vertices. Nikiforov's own impressions on the project are perhaps a good characterization of the value of graph algorithms using spectral methods – the development of tools for reframing traditional graph theoretic results in a simple way.

2 Spectral Properties of Graph Matrices

We begin with some basic graph theoretic definitions that will be used throughout the report. We also introduce our key players, the Laplacian matrices.

Let $G = (V, E)$ be a simple graph on n vertices and m edges and let the vertices of V be arbitrarily indexed by an index set $\mathcal{I} = \{1, 2, \ldots, n\}$. We will sometimes use a weighted graph, in which case $G = (V, E_w)$. We will be explicit in these cases. The number of vertices adjacent to a given vertex v_i is the *degree* of v_i, denoted $d_i = |\{v_j \in V | \{i, j\} \in E\}|$. Degree sequences of graphs have important implications in the diffusive and connective properties of the graph, which motivate the following definitions concerning graphs induced by subsets of vertices.

For a subset S, we refer to the edges whose removal separate S from the rest of the graph as the *edge boundary*, $\partial(S)$, of S,

$$\partial(S) = \{i, j \in E | v_i \in S, v_j \in \bar{S}\}.$$

Similarly, the *vertex boundary*, $\delta(S)$ is the set of vertices not in S which border S,

$$\delta(S) = \{v_i \in \bar{S} | \{i, j\} \in E \text{ for some } v_j \in S\}.$$

Here, we use $\bar{S}$ to denote the complement of S in V.

2.1 Implications of graph matrices

Spectral graph theory began as an investigation of the spectral properties of basic graph matrices. Here we discuss these matrices and some properties of interest. As a note on notation, if M is a graph matrix, we say $G(M)$ is a graph associated to the matrix; i.e. $G(M)$ is a graph for which the graph matrix of G is M by some permutation of its vertices.

The Adjacency Matrix Many are familiar with the adjacency matrix, the $n \times n$ symmetric matrix A defined by

$$(A)_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, A is real, symmetric, and nonnegative. This implies a few nice properties about the eigensystem of the adjacency matrix:

1. A has n real eigenvectors, $u_1, \ldots, u_n$ and eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$.
2. The eigenvectors of A can be chosen to be orthonormal (that is, pairwise distinct vectors are orthogonal and each vector has unit length).
3. A is irreducible if and only if $G(A)$ is connected.
4. The *spectral radius* of a graph is the eigenvalue of greatest magnitude, and this is bounded by the minimum and maximum degrees of the vertices of the graph.
5. $A = \sum_{i=1}^{n} \lambda_i u_k u_i^T$.

An early graph partitioning algorithm due to Barnes [12] uses the eigenvectors of A. In particular, he uses a spectral analysis to show that the partitioning problem can be reduced to the problem of approximating the adjacency matrix by the partition indicator matrix, P, defined by $(P)_{ij} = 1$ if v_i and v_j belong to the same subset. This result is related to the partitioning algorithm of [28] with two major changes. First, the algorithm of [28] uses a matrix related to the Laplacian L, discussed below. Second, Barnes presents something of an extention to multiway partitions by using $k-1$ eigenvectors for a k-way partition.

The adjacency matrix has also been used for alternative solutions to classical combinatorial problems. Aspvall and Gilbert, for example, use point 5 to show how the eigenvalue decomposition of the adjacency matrix can be used to color a k-partite graph in their graph coloring heuristic [10]. Nikiforov used the eigenvectors of A to prove extremal properties [46, 47], and Grimes et al. [35] use an eigenvector of the variation $A' = A + I$ to find a pseudoperipheral vertex of a graph.

Laplacian Matrices Although adjacency matrices are useful, Laplacian matrices have by far more applications. A most general definition of a Laplacian matrix is given by Spielman, who has devoted much of his career to studying them. He defines Laplacian matrices of graphs to be symmetric, with zero row-sums and non-positive off-diagonal matrices, as there is always a graph associated to this Laplacian [53]. From the definition above, we see that one reason a Laplacian is so important is the more direct involvement of vertex degree in Laplacian equations. We make more explicit the most common forms and variations of the Laplacian.

- *The combinatorial Laplacian*

  Define D to be the diagonal degree matrix, where each diagonal element is the degree of the corresponding vertex, $(D)_{ii} = d_i$. This brings us to our first Laplacian matrix, $L = D - A$.

  Many spectral properties of L have been studied by [37, 30], and indeed this is perhaps the most prominent version of the Laplacian in the literature. One reason for this is its neat quadratic form:

  $$x^T L x = \sum_{i \sim j} (x(i) - x(j))^2.$$

  We use $i \sim j$ to denote the symmetric relationship $\{i,j\} \in E$. This form is convenient for modeling linear systems in which values on vertices are desired to converge to a common value, as in consensus algorithms [51]. This Laplacian can be used to enumerate spanning trees of a graph. This is of independent interest but is also often used as a subroutine for construction and traversal algorithms. Finally, it is symmetric, singular, and positive semidefinite.

Let E_o be the edge set with an arbitrary but fixed orientation for each edge. Then we define the $n \times m$ edge-incidence matrix B

$$B(i,j) = \begin{cases} 1 & \text{if } v_i \text{ is the head of } e_j \text{ in } E_o, \\ -1 & \text{if } v_i \text{ is the tail of } e_j \text{ in } E_o, \\ 0 & \text{otherwise.} \end{cases}$$

and $L = BB^T$ regardless of the orientation choice of G (see, for instance, [21, 33]).

The Laplacian is closely associated to graph connectivity. First, it is irreducible if and only if the graph is connected. Second, the eigenvalues of L give useful bounds on the edge ratios of the graph $G(L)$ [30, 31]. In particular, the second smallest eigenvalue of L is referred to as the *algebraic connectivity* (or *Fiedler value*) of the graph because bounds on graph cuts can be derived from this value.

- *The Laplace operator*

  Consider the random walk where each step is taken to a neighboring vertex with uniform probability. This walk can be summarized by the transition probability matrix $(P)_{ij} = 1/d_i$ for any neighbor v_j of v_i, and $(P)_{ij} = 0$ otherwise. This is neatly summarized as $P = D^{-1}A$.

  We refer to $\Delta = I - D^{-1}A$ as the *discrete Laplace operator*. Although it is not symmetric, is useful because of its connection to random walks and the discrete Green's function, which is related to the heat kernel and will be discussed in more detail later.

- *The normalized Laplacian*

  Finally we consider the *normalized Laplacian* $\mathcal{L} = D^{-1/2}LD^{-1/2} = D^{1/2}\Delta D^{-1/2}$. The matrix $\mathcal{L}$ is the degree-normalized form of L, and a symmetric version of Δ. The normalized Laplacian can be viewed as an operator on the space of functions $f : V \to \mathbb{R}^n$ defined by

$$\mathcal{L}f(i) = \frac{1}{\sqrt{d_i}} \sum_{j:i\sim j} \Big(\frac{f(i)}{\sqrt{d_i}} - \frac{f(j)}{\sqrt{d_j}} \Big).$$

  Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ now be the eigenvalues of $\mathcal{L}$ and ϕ_i be the projection to the ith orthonormal eigenvectors. All of the eigenvalues of $\mathcal{L}$ are nonnegative, and $\lambda_1 = 0$. Then, when restricting to the space orthogonal to ϕ_1 corresponding to λ_1, we have that

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

  In particular, when S is a subset of vertices of $G(\mathcal{L})$ that induces a connected subgraph, we have

$$\mathcal{L}_S = \sum_{i=1}^{s} \lambda_i \phi_i, \tag{1}$$

  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 $\mathcal{L}_S$. 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 2.1. *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 $i+1$ 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.*

We note here that, although each matrix has its advantage, only L is both symmetric and diagonally dominant.

3 Solving Linear Systems in the Graph Laplacian

The problem of interest is an efficient way to approximate a solution to the system of equations $Lx = b$, where the matrix L is a graph Laplacian. Linear systems in the graph Laplacian arise in many natural contexts, including measuring the effective resistance of an electrical network, calculating a maximum flow, characterizing the motion of coupled oscillators, and finding consensus in multi-agent networks (see [23]).

Linear systems in the graph Laplacian may seem like a small class of problems. However, it is shown in [34] that any symmetric, diagonally dominant (SDD) system can be transformed into a Laplacian system in linear time. Similarly, we can easily transform linear systems in any the of above mentioned Laplacians. As an example, as long as the graph of our model is connected and the matrix $D(G)$ is invertible, a system in $\mathcal{L}$ can be deduced from a system in L,

$$Lx = b \quad \text{and} \quad \mathcal{L}x_1 = b_1$$

by setting $x_1 = D^{1/2}x$ and $b_1 = D^{-1/2}b$. Therefore, we may limit our discussion to Laplacians, and assume this transformation whenever discussing general SDD linear systems, $Ax = b$.

3.1 Historical methods

Of course, when computing the inverse A^{-1} is feasible, we may obtain an exact solution $x = A^{-1}b$ by a single matrix-vector multiplication in $O(n^2)$ time. In general, however, inverting a matrix is difficult. Gaussian elimination, for example, is a procedure for inverting the matrix in order to directly compute an exact solution, and takes $O(n^3)$ time.

In practice, then, two classes of algorithms which are based on Gaussian elimination are used for computing good approximations to the solution. First are iterative methods, which typically involve only matrix-vector multiplications or other simple vector operations. Iterative methods approximate a solution by successively testing and improving an estimated solution. In each iteration, a new vector $\hat{x}$ is computed for which $A\hat{x}$ is closer to b than the last. Commonly used iterative methods are the conjugate gradient and the Chebyshev.

The second class are the preconditioned, or inexact, iterative methods, in which an approximation of the matrix is instead used. This approximation is called the preconditioner. We consider a matrix B to be a good preconditioner of A when it is a very good approximation of A, can be computed quickly, reduces the number of iterations required, and further can be *solved* quickly. To clarify the last point, consider the preconditioned Richardson's method [61]:

$$\begin{aligned} x^{(0)} &= 0 \\ x^{(i+1)} &= B^{-1}b + (I - B^{-1}A)x^{(i)} \text{ for } i > 0. \end{aligned}$$

This yields a solution to the preconditioned system $B^{-1}Ax = B^{-1}b$. We see that the above involves a computation of $B^{-1}b$, which involves solving the linear system $By = b$ to avoid a direct inverse computation. Then, a good preconditioner is one which itself can be solved quickly, and ideally in linear time. This, in general, is very difficult, as any preconditioner B will not be significantly easier to solve than A.

This brings us to a third class of linear solvers. In recursive preconditioned methods, the system in B is only solved approximately with a recursive invocation of the same iterative method. This means finding a preconditioner for B, and a preconditioner for that preconditioner, recursively. Recursive preconditioned methods were introduced by Spielman and Teng in [55]. They later apply this linear solver with a quality preconditioning algorithm to present the first SDD solver that runs in time $O(m \log^c n)$ for a large constant c [56]. Their solver is the subject of the coming sections.

4 The First Nearly-Linear Time Solver

The recursive solver of Spielman and Teng, from here on referred to as the ST-solver, broke complexity boundaries in a big way. Their linear solver is the first which computes solutions $\hat{x}$ to SDD systems

$Ax = b$ with $||\hat{x} - x|| \leq \epsilon$ in time $O(m \log^{O(1)} m)$, where the exponent is a large constant. The key to their algorithm is twofold. First is the innovative recursive procedure for preconditioning a matrix, as mentioned above. Second is the way in which they compute the preconditioners.

4.1 The quality of a preconditioner

As mentioned above, one criterion for a good preconditioner is to be a good approximation of the original matrix. This necessitates an approximation measure. For this, we define the operator $A \preceq B$ which holds when

$$x^T Ax \leq x^T Bx$$

for all vectors x. It has been shown that, if A and B are positive semidefinite matrices with the same nullspace, then all the eigenvalues λ_i of $AB^\dagger$ lie in the range $\kappa_{\min} \leq \lambda_i \leq \kappa_{\max}$ if and only if $\kappa_{\min} B \preceq A \preceq \kappa_{\max} B$ (see [57]). [1]

Define the *generalized condition number*, $\kappa(A, B)$, to the be ratio of the largest to smallest nonzero eigenvalues of $AB^\dagger$. Then it follows that $\kappa_{\min} B \preceq A \preceq \kappa_{\max} B$ implies that $\kappa(A, B) \leq \kappa_{\max}/\kappa_{\min}$. This value bounds the number of iterations required in the preconditioned iterative method, so we can view it as a version of the condition number of a matrix. [2] Then, we say that B is a *κ-approximation* of A for symmetric matrices A and B when

$$\kappa_{\min} B \preceq A \preceq \kappa_{\max} B.$$

Alternately, a graph H is a κ-approximation of a graph G if

$$\kappa_{\min} L(H) \preceq L(G) \preceq \kappa_{\max} L(H)$$

where $\kappa = \kappa_{\max}/\kappa_{\min}$.

4.2 Using recursion to precondition

The preconditioned Richardson's method is an example of a one-level iterative solver, in which a single preconditioner of the input matrix is used for the iterative procedure. Spielman and Teng expand upon this idea with a *recursive* solver. In [56] and the expanded version [57], the ST-solver implements the following idea:

Idea. *To solve linear systems in A, compute a preconditioner B as in one-level iterative methods. However, at this point systems in B are still likely to be difficult to solve. Reduce the matrix B to a matrix A_1, a refined version of A, and recursively solve the system in A_1.*

This process yields a chain of matrices $\mathcal{C} = \{A, B, A_1, B_1, \ldots, A_l\}$ until A_l is small enough in both dimension and number of nonzero entries. That is, each matrix B_i is a preconditioner for A_i, and then rather than solving systems in B_i directly, these are reduced to systems in A_{i+1}. The Preconditioned Chebyshev method is used at the base of the recursion.

The chain-building procedure makes use of two distinct preconditioning routines, that which reduces the A_i to the B_i, and that which reduces the B_i to the A_{i+1}. Although these result in modified matrices, we discuss the outcome on the graphs of the matrices, i.e. the chain $\mathcal{C} = \{G, H, G_1, H_1, \ldots, G_l\}$ of progressively smaller graphs, for reasons which will become clear. To obtain G_{i+1} from H_i, a partial Cholesky factorization is used to greedily remove vertices of degree one or two. The other procedure, for preconditioning G_i, uses a spectral sparsification method that warrants deeper explanation.

[1] $B^\dagger$ is the Moore-Penrose pseudoinverse of B, the matrix with the same nullspaces as B and acts as the inverse of B on its image.

[2] A general definition of the condition number of a matrix is $\kappa(A) = ||A^{-1}|| \cdot ||A||$. When the matrix norm is the 2-norm, it follows that $\kappa_2(A) = \lambda_{\max}/\lambda_{\min}$, where these are eigenvalues of the matrix.

4.3 Spanning trees for sparsification

Spielman and Teng design a spectral sparsifier that is inspired by the insight of Vaidya [59] that subgraphs serve as good graph preconditioners. Vaidya uses a maximum weight spanning tree of the graph as the base for the preconditioning subgraph and adds t^2 edges to obtain a sparse graph that $O(nm/t^2)$-approximates A.

The ST-solver similarly uses a spanning tree as a base for the preconditioners, but not the maximum weight spanning tree of Vaidya. For their base tree, Spielman and Teng instead consider the *stretch* of edges, defined by the following. Let T be a spanning tree of a weighted graph G, and consider an edge e in G of weight $w(e)$. Define $w'(e) = 1/w(e)$, the reciprocal of the edge weight. Then if the unique path in T connecting endpoints of e consists of tree edges $e_1, e_1, \ldots, e_k$, define the stretch of e by T to be

$$stretch_T(e) = \frac{\sum_{i=1}^{k} w'(e_i)}{w'(e)}.$$

This value can be interpreted as the cost of the detour forced by traversing T instead of G. The total stretch of a graph with respect to a tree T is the sum of the stretch by T of all off-tree edges.

The ST-solver uses the low-stretch spanning tree (LSST) construction of Alon, Karp, Peleg, and West [5] as a base for preconditioners. After computing an LSST, edges are then reweighted and added back to the tree through a random sampling in which the probability of drawing an edge is related to vertex degree.

The ST-solver was the fastest-known solver for a number of years. The next improvements came from a modification of the sparsifier using a notion of graph distance. We discuss the solver of Koutis, Miller, and Peng which makes novel use of this idea.

5 Improving the ST-Solver: Sparsification by Effective Resistances

The biggest improvement made from the ST-solver was that of Koutis et al. at first in [41] and later improved in [42]. Their algorithm is quite similar to that of Spielman and Teng, but with an alternative sparsification procedure which yields a better chain of preconditioners for the recursive solver and requires fewer iterations. This modification improved the expected runtime to $O(m \log^2 n \log(1/\epsilon))$ for solutions with ϵ approximation guarantees.

The algorithm of Koutis et al. again uses an LSST for the base of the graph sparsifier, and replaces off-tree edges through a random sampling. The difference is the way probabilities are assigned for the random sampling; in this case the probabilities are according to the effective resistance of the edge. It turns out that these probabilities yield sparsifiers with few edges.

An obvious concern here is that computing effective resistances themselves involve solving a linear system. This circular connundrum is circumvented by a simple modification that uses an upper estimate of the effective resistance of each edge rather than an exact value. Specifically, their sampling procedure, named `Sample`, chooses edges with probabilities according to

$$p'_e \geq w_e R_e,$$

where w_e is the weight of the edge in the graph, and R_e is the effective resistance of the edge computed with edge conductances assigned $1/w_e$. The graph H returned by the sampling procedure `Sample` satisfies $L(G) \preceq 2L(H) \preceq 3L(G)$ with high probability.

The sampler is adapted from the the sparsifier of Spielman and Srivastava [54], where an LSST is augmented by drawing edges according to the probabilities $p'_e = w_e R_e$.

To summarize, the incremental sparisification procedure first computes an LSST of the original graph with total stretch $O(m \log n)$ using the algorithm modified from [54]. This is called by `LowStretchTree`. The edge weights are then scaled by a constant factor k so the total stretch is now $O(m \log(n/k))$. Add this scaled version to the sparsifier. Effective resistance estimates are computed using only the edges

of the modified LSST. Edges are then added back by oversampling with edge probabilities proportional to their stretch over the scaled tree, taking $\tilde{O}(m \log^2(n/k))$ samples. We give the incremental sparsifier below.

`IncrementalSparsify`(G, k, ξ):

$T \leftarrow$ `LowStretchTree`(G)
Let T' be T scaled by a factor of k
Let G' be the graph obtained from G by replacing T by T'
for $e \in E$ **do**
  Calculate $stretch_{T'}(e)$
end for
$H \leftarrow$ `Sample`$(G', stretch_{T'}, 1/2\xi)$ **return** $2H$

The algorithm `IncrementalSparsify` outputs a graph on $n-1+m/k$ edges that $\tilde{O}(k \log^2 n)$-approximates the graph with probability $1 - \xi$.

The final result of [41] is a recursive linear solver that computes a vector $\hat{x}$ which satisfies $||\hat{x} - A^\dagger b||_A < \epsilon ||A^\dagger b||_A$ in expected time $O(m \log^2 n \log(1/\epsilon))$. The algorithm is improved in [42] due to a substantial cut in the number of LSST computations. The governing insight is that the total stretch of off-tree edges is invariant under sparsification. That is, the total stretch of off-tree edges of H_i is at most equal to G_{i-1}, and the total stretch is conserved under the graph contraction process. The key improvement in the algorithm of [42], then, comes from computing only a single LSST for the first graph in the chain, rather than at each step. The result of this improvement is an algorithm that computes an approximate solution to $Ax = b$ in time $\tilde{O}(m \log n \log(1/\epsilon))^2$.

Recursive preconditioning solvers constitute a new class of linear solvers that has proven to be highly competitive in efficiency and progressively provide better and better approximations of the true solutions. In the next section we discuss a new linear solver which is not based on iterations, but random walks. This translates to a considerable reduction in computation, and achieves a sublinear runtime.

6 Random Walks for Solving Linear Systems

Recall the problem at hand is finding a good approximation of the vector x which is a solution to the system of equations $Lx = b$. Let S be the subset of vertices of $G(L)$ given by the vector b,

$$S = \{v_i \in V |\ b[i] = 0\},$$

and say that $|S| = s$. Then the given vector b can be viewed as a vector of boundary conditions. Since L is an irreducible matrix exactly when $G(L)$ is connected, the nullspace of L is spanned by constant vectors. This means that $Lx = b$ will have a solution only if the sum of the entries of b equals zero. In many situations, the support of b will be quite small, and most of the $b[i]$ will be zero. In the example of an electrical network, for instance, the support of b consisted of exactly two vertices, the point of injection and the point of extraction, which held values 1 and -1 respectively.

We say the vector x satisfies the boundary condition when $x[i] = b[i]$ for all v_i in the support of b. Then the problem of solving the linear system $Lx = b$ with given boundary conditions is that of finding a vector x such that

$$x[i] = \begin{cases} \frac{1}{d_i} \sum_{j:i\sim j} x[j] & \text{if } v_i \in S, \\ b[i] & \text{if } v_i \notin S. \end{cases}$$

By examining this further, we see that the solution should satisfy

$$D_S x = A_S x + A_{\delta S} b$$
$$x_S = (D_S - A_S)^{-1}(A_{\delta S} b).$$

Here, we use D_S and A_S to be the matrices D and A, respectively, with rows and columns restricted to the vertices of S and x_S is the vector x over S. Similarly, $A_{\delta S}$ is the $s \times |\delta(S)|$ matrix with columns

restricted to $\delta(S)$ and rows to S. The matrix $(D_S - A_S)$ is, of course, the is the familiar L with rows and columns similarly restricted. When the induced subgraph of S is connected and $\delta(S)$ is nonempty, it is known that $(D_S - A_S)^{-1} = L_S^{-1}$ exists (see [21]). Then we have that the solution x to the system $Lx = b$ satisying the boundary condition b, when restricted to S, can be computed by

$$x_S = L_S^{-1}(A_{\delta S} b).$$

This can be reformulated in the normalized Laplacian $\mathcal{L}$ as

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

So far this does not seem to achieve much, as we are still faced with a matrix inversion. However, Chung and Simpson recently demonstrate in [23] that the vector x_S can be efficiently approximated without computing the inverse. The key is the heat kernel pagerank vector, which can be summarized as an exponential sum of random walks on the graph.

6.1 Heat kernel and random walks

The *Dirichlet heat kernel* restricted to a subset S on a graph is determined by a parameter t, interpreted as the temperature, and is defined by

$$\mathcal{H}_{S,t} = e^{-t\mathcal{L}_S} = D_S^{1/2} e^{-t\Delta_S} D_S^{-1/2}.$$

Let $\mathcal{G}$ denote the inverse of $\mathcal{L}_S$. Then, using (1), we have that

$$\mathcal{L}_S^{-1} = \mathcal{G} = \sum_{i=1}^{s} \frac{1}{\lambda_i} \phi_i.$$

It is known ([24, 23]) that $\mathcal{G}$ is related to $\mathcal{H}_{S,t}$ by

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

We remind the reader that L_S^{-1} is related to $\mathcal{L}_S^{-1}$ with a simple transformation and it becomes clear that computing $L_S^{-1} b$ is related to

$$\mathcal{L}_S^{-1} b = \mathcal{G} b = \int_0^\infty \mathcal{H}_{S,t} b \, \mathrm{d}t.$$

In [23], Chung and Simpson give a new way of computing this integral by a series of good approximations. First, the indefinite integral is taken to a finite T, $\int_0^T \mathcal{H}_{S,t} b \, \mathrm{d}t$, such that the tail end of the integral in negligibly small. Next, the integral is approximated by a finite Riemann sum,

$$\int_0^T \mathcal{H}_{S,t} b \, \mathrm{d}t \approx \sum_{j=1}^{N} b^T \mathcal{H}_{S,jT/N} \cdot \frac{T}{N}$$

for an appropriately large N. Here we make use of the fact that $\mathcal{H}$ is symmetric. Finally, the sum is approximated by taking random samples of $b^T \mathcal{H}_{S,t}$ for enough values of t. Thus, the problem is reduced to the problem of efficiently computing $b^T \mathcal{H}_{S,t}$.

The *Dirichlet heat kernel pagerank* is the vector

$$\rho_{t,f} = f^T e^{-t\Delta_S} = e^{-t} \sum_{k=0}^{\infty} \frac{t^k}{k!} f^T P_S^k,$$

where f is a vector of size s and again P_S is the transition probability matrix $P = D^{-1}A$ restricted to S. That is, $\rho_{t,f}$ is the exponential sum of random walks over the graph with starting distribution derived from the vector f. If we define $H_{S,t} = D^{-1/2} \mathcal{H}_{S,t} D^{1/2}$, such that $\mathcal{H}_{S,t}$ is the symmetric version of $H_{S,t}$, we have that

$$\rho_{t,f} = f^T H_{S,t}.$$

With some degree normalization, the samples of $b^T \mathcal{H}_{S,t}$ can then approximated by a heat kernel pagerank vector.

6.2 Approximating the heat kernel and solving the system

The linear solver of Chung and Simpson approximates the matrix-vector product $x_S = \mathcal{L}_S^{-1} b$ by a number of heat kernel pagerank approximations, and the runtime of their solver is dominated by the approximation algorithm. As mentioned, the vector $\rho_{t,f}$ is an exponential sum of random walks. Then, a clear way to approximate this is by a finite number of truncated random walks which avoid losing information from later walk steps as best as possible.

Below we present the algorithm `ApproxHK` which takes as input a graph G, a subset of vertices S, an arbitrary vector f of size s, a real value t and an error factor ϵ for approximating a heat kernel pagerank vector $\rho_{t,f}$.

`ApproxHK`(G, f, S, t, ϵ)

initialize 0-vector y of dimension s, where $s = |S|$.
$f_+ \leftarrow$ the positive portion of f
$f_- \leftarrow$ the negative portion of f so that $f = f_+ - f_-$
$h_1 \leftarrow \|D^{1/2} f_+\|_1$ and $h_2 \leftarrow \|D^{1/2} f_-\|_1$, where $\|\cdot\|_1$ indicates the L_1-norm.
$r \leftarrow \frac{16}{\epsilon^3} \log s$
$K \leftarrow \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}$
for r iterations **do**
  choose a starting node v_1 according to the distribution vector $f_+ = D^{1/2} f_+ / h_1$.
  Start
    simulate a $P_S = D_S^{-1} A_S$ random walk where k steps are taken with probability $e^{-t} \frac{t^k}{k!}$ where $k \leq K$, let u_1 be the last node visited in the walk
    $y[u_1] \leftarrow y[u_1] + h_1 / \sqrt{d_{u_1}}$
  End
  choose a starting node v_2 according to the distribution vector $f_- = D^{1/2} f_- / h_2$.
  Start
    simulate a $P_S = D_S^{-1} A_S$ random walk where k steps are taken with probability $e^{-t} \frac{t^k}{k!}$ where $k \leq K$, let u_2 be the last node visited in the walk
    $y[u_2] \leftarrow y[u_2] - h_2 / \sqrt{d_{u_2}}$
  End
end for
return $1/r \cdot y$

The heart of the algorithm lies within the **Start** and **End** limits, where the random walk is simulated. With the assumption that a unit time allows a random walk step and sampling from a distribution, the algorithm `ApproxHK` returns an approximate heat kernel pagerank vector within a factor of $1 + \epsilon$ in time $O\left(\frac{\log(1/\epsilon) \log s}{\epsilon^3 \log\log(1/\epsilon)}\right)$.

The linear solver of [23] makes a bounded number of call to the `ApproxHK` approximation procedure to compute a solution to the system $\mathcal{L}x = b$ in sublinear time. The algorithm called `GreensSolver`[3] summarizes the approach. Theorem 6.1 is the major result.

Theorem 6.1. *Let G be a graph and $\mathcal{L}$ the Laplacian of G. For the linear system $\mathcal{L}x = b$, the solution x is required to satisfy the boundary condition b, i.e., $x[i] = b[i]$ for $v_i \in support(b)$. Let $S = V \setminus support(b)$ and $s = |S|$. Suppose the induced subgraph on S is connected and $support(b)$ is nonempty. Then the approximate solution $\hat{x}$ output by the algorithm `GreensSolver`(G, b, ϵ) satisfies the following:*

1. *The absolute error of $\hat{x}$ is*
$$||x - \hat{x}|| \leq O(\epsilon(1 + ||b||))$$
with probability at least $1 - \epsilon$.
2. *The running time is $O\left(\frac{(\log s)^2 (\log(\epsilon^{-1}))^2}{\epsilon^5 \log\log(\epsilon^{-1})}\right)$ with additional preprocessing time $O(|\partial(S)|)$.*

[3]The name GreensSolver comes from $\mathcal{G}$, the Green's function for a graph.

`GreensSolver`(G, b, ϵ):

initialize a vector x of size $s = |S| = |V \setminus support(b)|$ of all zeros
$b_1 \leftarrow D_S^{-1/2} A_{\delta S} D_{\delta S^{-1/2}} b$
$T \leftarrow s^3 \log(1/\epsilon)$
$N \leftarrow T/\epsilon$
$r \leftarrow \epsilon^{-2}(\log s + \log(1/\epsilon))$
for $j = 1$ to r **do**
  $x_j \leftarrow$ `ApproxHK`$(G, b_1, S, kT/N, \epsilon)$ where k is chosen uniformly from $[1, N]$
  $x \leftarrow x + x_j$
end for
return x/r

The runtime of this algorithm is better than the iterative and recursive solvers mentioned earlier. The algorithm also introduces a new approach for solving linear systems which does not use iterative improvements and requires only a number of vector approximations which can be achieved by simulating random walks for a bounded number of steps. The vector approximation algorithm is reponsible for the runtime, but is also of independent interest, following in a line of efficient pagerank approximation methods [9, 25, 18].

6.3 Coda: Other Competitive Solvers

Since the ST-solver demonstrated that nearly-linear time algorithms for solving SDD systems are possible, some alternative approaches other than those mentioned have been developed for algorithms that meet and beat this bound. We briefly mention some of the best.

The combinatorial algorithm of Kelner et al. in [39] is distinct from those already discussed. Their algorithm works by repeatedly applying a non-recursive update rule to a spanning tree of the graph and runs in time $O(m \log^2 n \log\log n \log(1/\epsilon))$ in the unit-cost RAM model.

In [43], Lee and Sidford develop a way to accelerate randomized coordinate descent methods which both limit the cost per iteration and achieve faster convergence. With this method they achieve an $O(m \log^{3/2} n \sqrt{\log\log n} \log\log(n/\epsilon))$ runtime in the unit-cost RAM model for solving SDD linear systems.

7 The Laplacian Paradigm

With their rapid development, spectral algorithms for massive graphs are being viewed more and more as algorithmic primitives. The success of the ST-solver especially won the problem of solving sytems in the graph Laplacian a place among these, and Teng calls this problem the *Laplacian Primitive* [58]. The Laplacian Paradigm refers to a new class of algorithms, across different areas, which take advantage of these Laplacian solvers for improving efficiency. The thesis of the Laplacian Paradigm is that the Laplacian Primitive can be a powerful primitive for different combinatorial optimization and numerical analysis problems. We have already seen in Section 2.1 how spectral methods have been applied to classic combinatorial graph problems. The mission of the Laplacian Paradigm is that the Laplacian Primitive be used more for classical problems, in hopes that the impressive time complexity can improve graph theoretic problems to be solved in nearly-linear time.

7.1 A suite of primitives

Below are a few examples of other primitives in the suite of nearly-linear time algorithms.

Finding balanced partitions The graph Laplacian plays an important role in finding balanced cuts in graphs. An early major result in this area is due to Donath and Hoffman [28], which bounds the number of cut edges in terms of the eigenvalues of the Laplacian. An important consequence of this is a spectral bipartitioning heuristic which imposes a linear ordering on the vertices of a graph by the eigenvector associated to the algebraic connectivity. This procedure is often called a *sweep*, and operates

as follows. Let the second eigenvector ϕ_2 define a function on the vertices of $G(L)$. Then a partition is found by ordering the vertices of G by their ϕ_2 value and drawing a partition at some point in this ordering.

Multiway partitions can be found through extensions of this heuristic. A common adaptation is to simply iterate the above procedure on each partitioned section. This is controlled in [38] by imposing a terminating cut ratio. An alternative approach which performs somewhat better in practice is to instead use the top k eigenvectors to form a k-way partition [12, 19]. Another common practice is to transform a graph cutting problem to a clustering problem by projecting the vertices of the graph to points in d-space [7, 8].

The above heuristics typically do not perform better than $\tilde{O}(n^3)$. Some of the greatest success for spectral partitioning algorithms has been found in local partitioning algorithms, and here we achieve the nearly-linear time bounds.

The partitioning algorithm of Spielman and Teng [56] is based on a distribution of random walks. The algorithm is a *local* algorithm because it finds a balanced partition while only investigating the vertices on one side of the partition. It is common in local algorithms for the runtime to be in terms of the size of a portion of the graph. Let $\Phi(S)$ be the Cheeger ratio of a subset of vertices S with respect to a graph G,

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

The volume of a set of vertices is the sum of the degrees, given by $\mathrm{vol}(S) = \sum_{v_i \in S} d_i$.

The Cheeger ratio is a good measure of balance. Given a graph and a subset of vertices S such that $\Phi(S) < \phi$ and $\mathrm{vol}(S) \leq \mathrm{vol}(V)/2$, the local partitioning algorithm of Spielman and Teng finds a set of vertices T such that $\mathrm{vol}(T) \geq \mathrm{vol}(S)/2$ and $\Phi(T) < O(\phi^{1/3} \log^{O(1)} n)$ in time $O(m(\log n/\phi)^{O(1)})$.

Andersen et al. give an improved local partitioning algorithm in [9] using an approximation of the PageRank vector. This also uses a distribution of random walks of the graph. Their algorithm finds a set of Cheeger ratio $O(\sqrt{\phi \log k})$ given any set C with Cheeger ratio ϕ and volume k in time $O(m \log^4 m/\phi^2)$.

Spectral sparsification We illustrated how sparsifiers are crucial to the quality and efficiency of linear solvers. Largely due to the success of these linear solvers, efficient methods for sparsifying graphs stand as their own problem and are part of the suite of Laplacian primitives.

The major revelationary breakthrough of the sparsifier used in the ST-solver is the notion of the *spectral* similarity of graphs. Benczúr and Karger [13] define such a notion with a nonuniform sampling process for producing a sparsifier which preserves cut values. Their sparsification requires, for every set of vertices, that the weight of edges leaving the set be approximately the same in the original graph as in the sparsifier.

Chung and Zhao [25] give a successful sparsifier. In this algorithm, the graph is sparsified by sampling $O(1/2(n \log n))$ edges according to a value related to PageRank which can be computed in time $O(1/\beta(2+\beta)m \log(1/\epsilon))$ for error bounds β and ϵ. Of course, the sparsifier of Spielman and Srivastava [54], already mentioned, uses the effective resistances of edges to sparsify a graph. Their algorithm computes a sparsified weighted subgraph with $O(n \log(n/\epsilon)^2)$ for an additive error parameter ϵ. The sparsifier can be computed in time $O(m)$ but requires solving $O(\log n)$ linear systems to compute the effective resistances.

Low stretch spanning trees The LSST construction of the ST-solver was inspired by that of Alon, Karp, Peleg, and West in [5]. Their preconditioner base is a tree constructed by a graph theoretic game in which players alternate between choosing a spanning tree and choosing an edge. The payoff to the edge player is

- $\mathrm{cost}(T, e) = 0$ if the edge lies in the tree chosen by the player, and
- $\mathrm{cost}(T, e) = \mathit{cycle}(T, e)/w_e$ otherwise,

where $\mathit{cycle}(T, e)$ is the weight of the unique cycle formed when e is added to T. Alon et al. show that the value of this game is bounded by $e^{O(\sqrt{\log n \log\log n})}$. The result is an algorithm for constructing a spanning tree T of a graph G such that the average $\mathrm{cost}(T, e)$ over all edges is bounded above by $e^{O(\sqrt{\log n \log\log n})}$.

Boman and Hendrickson later contribute a body of work establishing a theoretic foundation for using spanning trees as preconditioners [16, 17]. In [29] Elkin et al. give an $O(m \log^2 n)$-time algorithm for computing spanning trees with total stretch $\tilde{O}(m \log^2 n)$. Abraham, Bartal, and Neiman [1] give an $O(m \log n + n \log^2 n)$ algorithm for producing a spanning tree with total stretch $\tilde{O}(m \log n)$ using star decomposition, and later Abraham and Neiman achieve a slightly improved result using petal decomposition [2].

7.2 The Laplacian Primitive

A body of work using the Laplacian primitive to achieve near-linear runtimes is already developing. We give two examples.

The graph learning problem takes as input a strongly connected directed graph, a subset of vertices, and a labeling function y that assigns to each vertex in the subset a label from $Y = \{1, -1\}$, and to each vertex outside the subset the label 0. If $\mathcal{H}(\mathcal{V})$ is the set of functions from $V \to \mathbb{R}$, the objective of the graph learning problem is to find a function $f \in \mathcal{H}(\mathcal{V})$ that minimizes

$$\Omega(f) + \mu ||f - y||^2,$$

for a constant parameter μ and where

$$\Omega(f) = \frac{1}{2} \sum_{i \sim j} \pi[i] p[i,j] \Big(\frac{f[i]}{\sqrt{\pi[i]}} - \frac{f[j]}{\sqrt{\pi[j]}} \Big).$$

We let π denote the stationary distribution of the random walk on the graph with transition probability given by p. Then, as a first example, Zhou, Huang, and Schölkopf show in [62] that the graph learning problem can be solved in nearly-linear time by solving a system of equations in the matrix

$$A = \Big(\Pi - \frac{1}{1+\mu} \frac{\Pi P + P^T \Pi}{2} \Big),$$

where Π is the diagonal matrix $(\Pi)_{ii} = \pi[i]$ and P is the directed version of our usual transition probability matrix.

Another application is approximating eigenvectors. The Fiedler vector is the second smallest eigenvector of L, and the Fiedler value is the corresponding eigenvalue, denoted here by $\lambda_2(L)$. An ϵ-approximate Fiedler vector v satisfies

$$\lambda_2(L) \leq \lambda(v) = \frac{v^T L v}{v^T v} \leq (1 + \epsilon) \lambda_2(L).$$

Spielman and Teng show how to apply the Laplacian Paradigm to the problem of computing an ϵ-approximate Fiedler vector. Choose a random unit vector r such that $v_1^T r = 0$, and v_1 is the eigenvector of L corresponding to the smallest eigenvalue of L, $\lambda_1 = 0$. Note that v_1 is the all-ones vector, and that r can be written $r = \sum_{i=2}^{n} c_i v_i$. Now, using the fact that

$$L^\dagger r = \sum_{i=1}^{n} c_i \lambda_i^{-1} v_i, \quad \text{and} \quad (L^\dagger)^t r = \sum_{i=1}^{n} c_i \lambda_i^{-t} v_i$$

in general for a positive integer $t \geq 1$, an ϵ-approximate Fiedler vector can be computed. This is because, if c_2 is not too small, by choosing $t = \Theta(\log(n/\epsilon)/\epsilon)$, the approximate vector can be computed using the inverse power method. With an efficient method for computing $L^\dagger r$, nearly-linear time is achieved. Details are given in [57].

8 Future Directions and Closing Remarks

It was demonstrated how the effective resistance of an edge serves as a good measure of similarity between the adjacent vertices. There are some critical properties of the heat kernel pagerank which suggest that

values over this vector may serve as a good similarity measures as well. As mentioned, a heat kernel pagerank vector is an exponential sum of random walks. When we use as the initial vector the indicator vector χ_i with all initial probability on vertex v_i, then the values of $\rho_{t,i} := \rho_{t,\chi_i}$ represent how likely a random walk starting from v_i is to end at a particular vertex. That is, the value $\rho_{t,i}[j]$ is a good indicator of how similar vertices v_i and v_j are from a diffusive standpoint.

This suggests an alternative approach to sparsification. The successful sparsifiers discussed in the survey add off-tree edges to a LSST with probabilities proportional to the effective resistance of the edge. What if the heat kernel value is used instead? The implications of this are compelling. First, it would provide a more meaningful measure for unweighted graphs. We remind the reader that the effective resistances were computed according to weights on edges. Second, the fast algorithm for computing heat kernel pagerank offers the possibility of impressive improvements in runtime, since there is no longer need to solve a linear system to compute vertex similarities. Beyond this, using heat kernel for edge values presents a myriad of applications including partitioning, clustering, anomaly detection and cooperation.

A main goal of the so-called Laplacian Paradigm is to advocate deeper research. Many of these algorithms are at a point of near-production, but would benefit from deeper investigation. This will involve a more careful analysis of the implications of the time/accuracy tradeoff, and an understanding of what would be acceptable in practice. It is in high hopes that this new suite of spectral algorithms inspires further research into exisiting problems using the Laplacian Primitives, and also into finding new applications. In particular, it is the author's hope that these linear solvers will make problems on massive graphs more tractable.

References

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman, *Nearly tight low stretch spanning trees*, Proceedings of the 49th annual Symposium on Foundations of Computer Science, IEEE, 2008, pp. 781–790.

[2] Ittai Abraham and Ofer Neiman, *Using petal-decompositions to build a low stretch spanning tree*, Proceedings of the 44th Symposium on Theory of Computing, ACM, 2012, pp. 395–406.

[3] Noga Alon, *Eigenvalues and expanders*, Combinatorica **6** (1986), no. 2, 83–96.

[4] Noga Alon, Zvi Galil, and Vitali D. Milman, *Better expanders and superconcentrators*, Journal of Algorithms **8** (1987), no. 3, 337–347.

[5] Noga Alon, Richard M. Karp, David Peleg, and Douglas West, *A graph-theoretic game and its application to the k-server problem*, SIAM Journal on Computing **24** (1995), no. 1, 78–100.

[6] Noga Alon and Vitali D. Milman, $\lambda_1, isoperimetric inequalities for graphs, and superconductors$, Journal of Combinatorial Theory, Series B **38** (1985), no. 1, 73–88.

[7] Charles J. Alpert and Andrew B. Kahng, *Multi-way partitioning via spacefilling curves and dynamic programming*, Proceedings of the 31st annual Design Automation Conference, ACM, 1994, pp. 652–657.

[8] Charles J. Alpert and So-Zen Yao, *Spectral partitioning: The more eigenvectors, the better*, Proceedings of the 32nd annual ACM/IEEE Design Automation Conference, ACM, 1995, pp. 195–200.

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

[10] Bengt Aspvall and John R. Gilbert, *Graph coloring using eigenvalue decomposition*, SIAM Journal on Algebraic Discrete Methods **5** (1984), no. 4, 526–538.

[11] Stephen T. Barnard, Alex Pothen, and Horst Simon, *A spectral algorithm for envelope reduction of sparse matrices*, Numerical Linear Algebra with Applications **2** (1995), no. 4, 317–334.

[12] Earl R. Barnes, *An algorithm for partitioning the nodes of a graph*, SIAM Journal on Algebraic Discrete Methods **3** (1982), no. 4, 541–550.

[13] András A Benczúr and David R Karger, *Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time*, Proceedings of the 28th annual ACM Symposium on Theory of Computing, ACM, 1996, pp. 47–55.

[14] Norman L. Bigg, Edward Keith Lloyd, and Robin James Wilson, *Graph theory: 1736-1936*, Oxford University Press, 1976.

[15] Béla Bollobás and Vladimir Nikiforov, *Cliques and the spectral radius*, Journal of Combinatorial Theory, Series B **97** (2007), no. 5, 859–865.

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

[17] ———, *Support theory for preconditioning*, SIAM Journal on Matrix Analysis and Applications **25** (2003), no. 3, 694–717.

[18] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Shang-Hua Teng, *A sublinear time algorithm for pagerank computations*, Algorithms and Models for the Web Graph, 2012, pp. 41–53.

[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** (1994), no. 9, 1088–1096.

[20] Jeff Cheeger, *A lower bound for the smallest eigenvalue of the laplacian*, Problems in Analysis **625** (1970), 195–199.

[21] Fan Chung, *Spectral graph theory*, American Mathematical Society, 1997.

[22] Fan Chung and Linyuan Lu, *Complex graphs and networks*, no. 107, American Mathematical Society, 2006.

[23] Fan Chung and Olivia Simpson, *Solving linear systems with boundary conditions using heat kernel pagerank*, 2013.

[24] Fan Chung and Shing-Tung Yau, *Discrete green's functions*, Journal of Combinatorial Theory, Series A **91** (2000), no. 1, 191–214.

[25] Fan Chung and Wenbo Zhao, *A sharp pagerank algorithm with applications to edge ranking and graph sparsification*, Algorithms and Models for the Web Graph, 2010, pp. 2–14.

[26] Lothar Von Collatz and Ulrich Sinogowitz, *Spektren endlicher grafen*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **21** (1957), no. 1, 63–77.

[27] Samuel I Daitch and Daniel A Spielman, *Faster approximate lossy generalized flow via interior point algorithms*, Proceedings of the 40th annual ACM symposium on Theory of computing, ACM, 2008, pp. 451–460.

[28] William E. Donath and Alan J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM Journal of Research and Development **17** (1973), no. 5, 420–425.

[29] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng, *Lower-stretch spanning trees*, Proceedings of the 37th annual Symposium on Theory of Computing, ACM, 2005, pp. 494–503.

[30] Miroslav Fiedler, *Algebraic connectivity of graphs*, Czech. Math. Journal **23** (1973), no. 98, 298–305.

[31] ———, *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory*, Czechoslovak Mathematical Journal **25** (1975), no. 4, 619–633.

[32] Miroslav Fiedler and Vladimir Nikiforov, *Spectral radius and hamiltonicity of graphs*, Linear Algebra and its Applications **432** (2010), no. 9, 2170–2173.

[33] Christoper David Godsil and Gordon Royle, *Algebraic graph theory*, Graduate Texts in Mathematics, vol. 207, Springer New York, 2001.

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

[35] Roger G. Grimes, Daniel J. Pierce, and Horst D. Simon, *A new algorithm for finding a pseudoperipheral node in a graph*, SIAM Journal on Matrix Analysis and Applications **11** (1990), no. 2, 323–334.

[36] Aric Hagberg and Daniel A Schult, *Rewiring networks for synchronization*, Chaos: An Interdisciplinary Journal of Nonlinear Science **18** (2008), no. 3, 037105–037105.

[37] William N. Anderson Jr. and Thomas D. Morley, *Eigenvalues of the laplacian of a graph**, Linear and Multilinear Algebra **18** (1985), no. 2, 141–145.

[38] Ravi Kannan, Santosh Vempala, and Adrian Vetta, *On clusterings: Good, bad and spectral*, Journal of the ACM (JACM) **51** (2004), no. 3, 497–515.

[39] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu, *A simple, combinatorial algorithm for solving sdd systems in nearly-linear time*, Proceedings of the 45th Annual ACM Symposium on Theory of Computing, ACM, 2013, pp. 911–920.

[40] Gustav Kirchhoff, *Über die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird*, Annalen der Physik **148** (1847), no. 12, 497–508.

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

[42] ———, *A nearly-$m \log n$ time solver for sdd linear systems*, IEEE 52nd Annual Symposium on Foundations of Computer Science, IEEE, 2011, pp. 590–598.

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

[44] Jure Leskovec, *Stanford network analysis project*, `http://snap.stanford.edu`.

[45] Bojan Mohar, *Isoperimetric numbers of graphs*, Journal of Combinatorial Theoryi, Series B **47** (1989), no. 3, 274–291.

[46] Vladimir Nikiforov, *Bounds on graph eigenvalues ii*, Linear Algebra and its Applications **427** (2007), no. 2, 183–189.

[47] ———, *A spectral condition for odd cycles in graphs*, Linear Algebra and its Applications **428** (2008), no. 7, 1492–1498.

[48] ———, *Some new results in extremal graph theory*, arXiv preprint (2011), arXiv:1107.1121.

[49] Vladimir Nikiforov and Richard H. Schelp, *Cycle lenghts in graphs with large minimum degree*, Journal of Graph Theory **52** (2006), no. 2, 157–170.

[50] Eva Nosal, *Eigenvalues of graphs*, Master's thesis, University of Calgary, 1970.

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

[52] Alex Pothen, Horst D. Simon, and Kang-Pu Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications **11** (1990), no. 3, 430–452.

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

[54] Daniel A. Spielman and Nikhil Srivastava, *Graph sparsification by effective resistances*, SIAM Journal on Computing **40** (2011), no. 6, 1913–1926.

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

[56] ———, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, Proceedings of the thirty-sixth annual ACM symposium on Theory of Computing, ACM, 2004, pp. 81–90.

[57] ———, *Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems*, CoRR **abs/cs/0607105** (2006).

[58] Shang-Hua Teng, *The laplacian paradigm: Emerging algorithms for massive graphs*, Theory and Applications of Models of Computation, Lecture Notes in Computer Science, vol. 6108, 2010, pp. 2–14.

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

[60] Virginia Vassilevska Williams, *Multiplying matrices faster than coppersmith-winograd*, Proceedings of the 44th Symposium on the Theory of Computing, 2012, pp. 887–898.

[61] David Young, *On richardson's method for solving linear systems with positive definite matrices*, Journal of Mathematics and Physics **32** (1950), 243–255.

[62] Denyong Zhou, Jiayuan Huang, and Bernhard Schólkopf, *Learning from labeled and unlabeled data on a directed graph*, Proceedings of the 22nd International Conference on Machine Learning, ACM, 2005, pp. 1036–1043.