

Algorithms, Graph Theory, and Linear Equations in Laplacian Matrices

Daniel A. Spielman *

Abstract. The Laplacian matrices of graphs are fundamental. In addition to facilitating the application of linear algebra to graph theory, they arise in many practical problems.

In this talk we survey recent progress on the design of provably fast algorithms for solving linear equations in the Laplacian matrices of graphs. These algorithms motivate and rely upon fascinating primitives in graph theory, including low-stretch spanning trees, graph sparsifiers, ultra-sparsifiers, and local graph clustering. These are all connected by a definition of what it means for one graph to approximate another. While this definition is dictated by Numerical Linear Algebra, it proves useful and natural from a graph theoretic perspective.

Mathematics Subject Classification (2000). Primary 68Q25; Secondary 65F08.

Keywords. keywords

1. Introduction

We all learn one way of solving linear equations when we first encounter linear algebra: Gaussian Elimination. In this survey, I will tell the story of some remarkable connections between algorithms, spectral graph theory, functional analysis and numerical linear algebra that arise in the search for asymptotically faster algorithms. I will only consider the problem of solving systems of linear equations in the Laplacian matrices of graphs. This is a very special case, but it is also a very interesting case. I begin by introducing the main characters in the story.

1. **Laplacian Matrices and Graphs.** We will consider weighted, undirected, simple graphs G given by a triple (V, E, w) , where V is a set of vertices, E is a set of edges, and w is a weight function that assigns a positive weight to every edge. The Laplacian matrix L of a graph is most naturally defined by the quadratic form it induces. For a vector $\mathbf{x} \in \mathbb{R}^V$, the Laplacian quadratic form of G is

$$\mathbf{x}^T L \mathbf{x} = \sum_{(u,v) \in E} w_{u,v} (\mathbf{x}(u) - \mathbf{x}(v))^2.$$

*This material is based upon work supported by the National Science Foundation under Grant Nos. 0634957 and 0915487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Thus, L provides a measure of the smoothness of \mathbf{x} over the edges in G . The more \mathbf{x} jumps over an edge, the larger the quadratic form becomes.

The Laplacian L also has a simple description as a matrix. Define the weighted degree of a vertex u by

$$d(u) = \sum_{v \in V} w_{u,v}.$$

Define D to be the diagonal matrix whose diagonal contains d , and define the weighted adjacency matrix of G by

$$A(u, v) = \begin{cases} w_{u,v} & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

We have

$$L = D - A.$$

It is often convenient to consider the *normalized Laplacian* of a graph instead of the Laplacian. It is given by $D^{-1/2}LD^{-1/2}$, and is more closely related to the behavior of random walks.

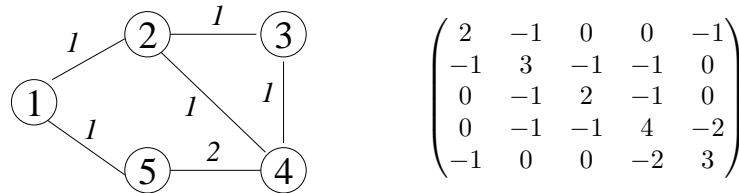


Figure 1. A Graph on five vertices and its Laplacian matrix. The weights of edges are indicated by the numbers next to them. All edges have weight 1, except for the edge between vertices 4 and 5 which has weight 2.

2. **Cuts in Graphs.** A large amount of algorithmic research is devoted to finding algorithms for partitioning the vertices and edges of graphs (see [LR99, ARV09, GW95, Kar00]). Given a set of vertices $S \subset V$, we define the boundary of S , written $\partial(S)$ to be the set of edges of G with exactly one vertex in S .

For a subset of vertices S , let $\chi_S \in \mathbb{R}^V$ denote the characteristic vector of S (one on S and zero outside). If all edge weights are 1, then $\chi_S^T L \chi_S$ equals the number of edges in $\partial(S)$. When the edges are weighted, it measures the sum of their weights.

Computer Scientists are often interested in finding the sets of vertices S that minimize or maximize the size of the boundary of S . In this survey, we will be interested in the sets of vertices that minimize the size of $\partial(S)$ divided

by a measure of the size of S . When we measure the number of vertices in S , we obtain the *isoperimetric number* of S ,

$$i(S) \stackrel{\text{def}}{=} \frac{|\partial(S)|}{\min(|S|, |V-S|)}.$$

If we instead measure the S by the weight of its edges, we obtain the *conductance* of S , which is given by

$$\phi(S) \stackrel{\text{def}}{=} \frac{w(\partial(S))}{\min(d(S), d(V-S))},$$

where $d(S)$ is the sum of the weighted degrees of vertices in the set S and $w(\partial(S))$ is the sum of the weights of the edges on the boundary of S . The isoperimetric number of a graph and the conductance of a graph are defined to be the minima of these quantities over subsets of vertices:

$$i_G \stackrel{\text{def}}{=} \min_{S \subset V} i(S) \quad \text{and} \quad \phi_G \stackrel{\text{def}}{=} \min_{S \subset V} \phi(S).$$

It is often useful to divide the vertices of a graph into two pieces by finding a set S of low isoperimetric number or conductance, and then partitioning the vertices according to whether or not they are in S .

3. **Expander Graphs.** Expander graphs are the regular, unweighted graphs having high isoperimetric number and conductance. Formally, a sequence of graphs is said to be a sequence of expander graphs if all of the graphs in the sequence are regular of the same degree and there exists a constant $\alpha > 0$ such that $\phi_G > \alpha$ for all graphs G in the family. The higher α , the better.

Expander graphs pop up all over Theoretical Computer Science (see [HLW06]), and are examples one should consider whenever thinking about graphs.

4. **Cheeger's Inequality.** The discrete versions of Cheeger's inequality [Che70] relate quantities like the isoperimetric number and the conductance of a graph to the eigenvalues of the Laplacian and the normalized Laplacian. The smallest eigenvalue of the Laplacian and the normalized Laplacian is always zero, and it has multiplicity 1 for a connected graph. The discrete versions of Cheeger's inequality (there are many, see [LS88, AM85, Alo86, Dod84, Var85, SJ89]) concern the smallest non-zero eigenvalue, which we denote λ_2 . For example, we will exploit the tight connection between conductance and the smallest non-zero eigenvalue of the normalized Laplacian:

$$2\phi_G \geq \lambda_2(D^{-1/2}LD^{-1/2}) \geq \phi_G^2/2.$$

The time required for a random walk on a graph to mix is essentially the reciprocal of $\lambda_2(D^{-1/2}LD^{-1/2})$. Sets of vertices of small conductance are obvious obstacles to rapid mixing. Cheeger's inequality tells us that they are the main obstacle. It also tells us that all non-zero eigenvalues of expander graphs are bounded away from zero. Indeed, expander graphs are often characterized by the gap between their Laplacian eigenvalues and zero.

5. **The Condition Number of a Matrix.** The *condition number* of a symmetric matrix, written $\kappa(A)$, is given by

$$\kappa(A) \stackrel{\text{def}}{=} \lambda_{\max}(A)/\lambda_{\min}(A),$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ denote the largest and smallest eigenvalues of A (for general matrices, we measure the singular values instead of the eigenvalues). For singular matrices, such as Laplacians, we instead measure the *finite condition number*, $\kappa_f(A)$, which is the ratio between the largest and smallest non-zero eigenvalues.

The condition number is a fundamental object of study in Numerical Linear Algebra. It tells us how much the solution to a system of equations in A can change when one perturbs A , and it may be used to bound the rate of convergence of iterative algorithms for solving linear equations in A . From Cheeger's inequality, we see that expander graphs are exactly the graphs whose Laplacian matrices have low condition number. Formally, families of expanders may be defined by the condition that there is an absolute constant c such that $\kappa_f(G) \leq c$ for all graphs in the family.

Spectrally speaking, the best expander graphs are the Ramanujan Graphs [LPS88, Mar88], which are d -regular graphs for which

$$\kappa_f(G) \leq \frac{d + 2\sqrt{d-1}}{d - 2\sqrt{d-1}}.$$

As d grows large, this bound quickly approaches 1.

6. **Random Matrix Theory.** Researchers in random matrix theory are particularly concerned with the singular values and eigenvalues of random matrices. Researchers in Computer Science often exploit results from this field, and study random matrices that are obtained by down-sampling other matrices [AM07, FK99]. We will be interested in the Laplacian matrices of randomly chosen subgraphs of a given graph.
7. **Spanning Trees.** A tree is a connected graph with no cycles. As trees are simple and easy to understand, it often proves useful to approximate a more complex graph by a tree (see [Bar96, Bar98, FRT04, ACF⁺04]). A spanning tree T of a graph G is a tree that connects all the vertices of G and whose edges are a subset of the edges of G . Many varieties of spanning trees are studied in Computer Science, including maximum-weight spanning trees, random spanning trees, shortest path trees, and low-stretch spanning trees. I find it amazing that spanning trees should have anything to do with solving systems of linear equations.

This survey begins with an explanation of where Laplacian matrices come from, and gives some reasons they appear in systems of linear equations. We then briefly explore some of the popular approaches to solving systems of linear equations, quickly jumping to preconditioned iterative methods. These methods solve linear

equations in a matrix A by multiplying vectors by A and solving linear equations in another matrix, called a *preconditioner*. These methods work well when the preconditioner is a good approximation for A and when linear equations in the preconditioner can be solved quickly. We will precondition Laplacian matrices of graphs by Laplacian matrices of other graphs (usually subgraphs), and will use tools from graph theory to reason about the quality of the approximations and the speed of the resulting linear equation solvers. In the end, we will see that linear equations in any Laplacian matrix can be solved to accuracy ϵ in time

$$O((m + n \log n (\log \log n)^2) \log \epsilon^{-1}),$$

if one allows polynomial time to precompute the preconditioners. Here n is the dimension and m is the number of non-zeros in the matrix. When m is much less than n^2 , this is less time than would be required to even read the inverse of a general n -by- n matrix.

The best balance we presently know between the complexity of computing the preconditioners and solving the linear equations yields an algorithm of complexity

$$O(m \log^c n \log 1/\epsilon),$$

for some large constant c . We hope this becomes a small constant, say 1 or 2, in the near future (In fact, it just did [KMP10]).

Highlights of this story include a definition of what it means to approximate one graph by another, a proof that every graph can be approximated by a sparse graph, an examination of which trees best approximate a given graph, and local algorithms for finding clusters of vertices in graphs.

2. Laplacian Matrices

Laplacian matrices of graphs are symmetric, have zero row-sums, and have non-positive off-diagonal entries. We call any matrix that satisfies these properties a Laplacian matrix, as there always exists some graph for which it is the Laplacian.

We now briefly list some applications in which the Laplacian matrices of graphs arise.

1. **Regression on Graphs.** Imagine that you have been told the value of a function \mathbf{f} on a subset W of the vertices of G , and wish to estimate the values of \mathbf{f} at the remaining vertices. Of course, this is not possible unless \mathbf{f} respects the graph structure in some way. One reasonable assumption is that the quadratic form in the Laplacian is small, in which case one may estimate \mathbf{f} by solving for the function $\mathbf{f} : V \rightarrow \mathbb{R}$ minimizing $\mathbf{f}^T L \mathbf{f}$ subject to \mathbf{f} taking the given values on W (see [ZGL03]). Alternatively, one could assume that the value of \mathbf{f} at every vertex v is the weighted average of \mathbf{f} at the neighbors of v , with the weights being proportional to the edge weights. In this case, one should minimize

$$\|D^{-1}L\mathbf{f}\|$$

subject to \mathbf{f} taking the given values on W . These problems inspire many uses of graph Laplacians in Machine Learning.

2. **Spectral Graph Theory.** In Spectral Graph Theory, one studies graphs by examining the eigenvalues and eigenvectors of matrices related to these graphs. Fiedler [Fie73] was the first to identify the importance of the eigenvalues and eigenvectors of the Laplacian matrix of a graph. The book of Chung [Chu97] is devoted to the Laplacian matrix and its normalized version.
3. **Solving Maximum Flow by Interior Point Algorithms.** The Maximum Flow and Minimum Cost Flow problems are specific linear programming problems that arise in the study of network flow. If one solves these linear programs by interior point algorithms, then the interior point algorithms will spend most of their time solving systems of linear equations that can be reduced to restricted Laplacian systems. We refer the reader who would like to learn more about these reductions to one of [DS08, FG07].
4. **Resistor Networks.** The Laplacian matrices of graphs arise when one models electrical flow in networks of resistors. The vertices of a graph correspond to points at which we may inject or remove current and at which we will measure potentials. The edges correspond to resistors, with the weight of an edge being the reciprocal of its resistance. If $\mathbf{p} \in \mathbb{R}^V$ denotes the vector of potentials and $\mathbf{i}_{ext} \in \mathbb{R}^V$ the vectors of currents entering and leaving vertices, then these satisfy the relation

$$L\mathbf{p} = \mathbf{i}_{ext}.$$

We exploit this formula to compute the effective resistance between pairs of vertices. The effective resistance between vertices u and v is the difference in potential one must impose between u and v to flow one unit of current from u to v . To measure this, we compute the vector \mathbf{p} for which $L\mathbf{p} = \mathbf{i}_{ext}$, where

$$\mathbf{i}_{ext}(x) = \begin{cases} 1 & \text{for } x = u, \\ -1 & \text{for } x = v, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

We then measure the difference between $\mathbf{p}(u)$ and $\mathbf{p}(v)$.

5. **Partial Differential Equations.** Laplacian matrices often arise when one discretizes partial differential equations. For example, the Laplacian matrices of path graphs naturally arise when one studies the modes of vibrations of a string. Another natural example appears when one applies the finite element method to solve Laplace's equation in the plane using a triangulation with no obtuse angles (see [Str86, Section 5.4]). Boman, Hendrickson and Vavasis [BHV08] have shown that the problem of solving general elliptic partial differential equations by the finite element method can be reduced to the problem of solving linear equations in restricted Laplacian matrices.

Many of these applications require the solution of linear equations in Laplacian matrices, or their restrictions. If the values at some vertices are restricted, then the problem in the remaining vertices becomes one of solving a linear equation in a diagonally dominant symmetric M -matrix. Such a matrix is called a Stieltjes matrix, and may be expressed as a Laplacian plus a non-negative diagonal matrix. A Laplacian is always positive semi-definite, and if one adds a non-negative non-zero diagonal matrix to the Laplacian of a connected graph, the result will always be positive definite. The problem of computing the smallest eigenvalues and corresponding eigenvectors of a Laplacian matrix is often solved by the repeated solution of linear equations in that matrix.

3. Solving Linear Equations in Laplacian Matrices

There are two major approaches to solving linear equations in Laplacian matrices. The first are direct methods. These are essentially variants of Gaussian elimination, and lead to exact solutions. The second are the iterative (indirect) methods. These provide successively better approximations to a system of linear equations, typically requiring a number of iterations proportional to $\log \epsilon^{-1}$ to achieve accuracy ϵ .

3.1. Direct Methods. When one applies Gaussian Elimination to a matrix A , one produces a factorization of A in the form LU where U is an upper-triangular matrix and L is a lower-triangular matrix with 1s on the diagonal. Such a factorization allows one to easily solve linear equations in a matrix A , as one can solve a linear equation in an upper- or lower-triangular matrix in time proportional to its number of non-zero entries. When solving equations in symmetric positive-definite matrices, one uses the more compact Cholesky factorization which has the form LL^T , where L is a lower-triangular matrix. If you are familiar with Gaussian elimination, then you can understand Cholesky factorization as doing the obvious elimination to preserve symmetry: every row-elimination is followed by the corresponding column-elimination. While Laplacian matrices are not positive-definite, one can use essentially the same algorithm if one stops when the remaining matrix has dimension 2.

When applying Cholesky factorization to positive definite matrices one does not have to permute rows or columns to avoid having pivots that are zero [GL81]. However, the choice of which row and column to eliminate can have a big impact on the running time of the algorithm. Formally speaking, the choice of an elimination ordering corresponds to the choice of a permutation matrix P for which we factor $PAP^T = LL^T$. By choosing an elimination ordering carefully, one can sometimes find a factorization of the form LL^T in which L is very sparse and can be computed quickly. For the Laplacian matrices of graphs, this process has a very clean graph theoretic interpretation. The rows and columns correspond to vertices. When one eliminates the row and column corresponding to a vertex, the resulting matrix is the Laplacian of a graph in which that vertex has been removed, but in which all

of its neighbors have been connected. The weights with which they are connected naturally depend upon the weights with which they were connected to the eliminated vertex. Thus, we see that the number of entries in L depends linearly on the sum of the degrees of vertices when they are eliminated, and the time to compute L depends upon the sum of the squares of the degrees of eliminated vertices.

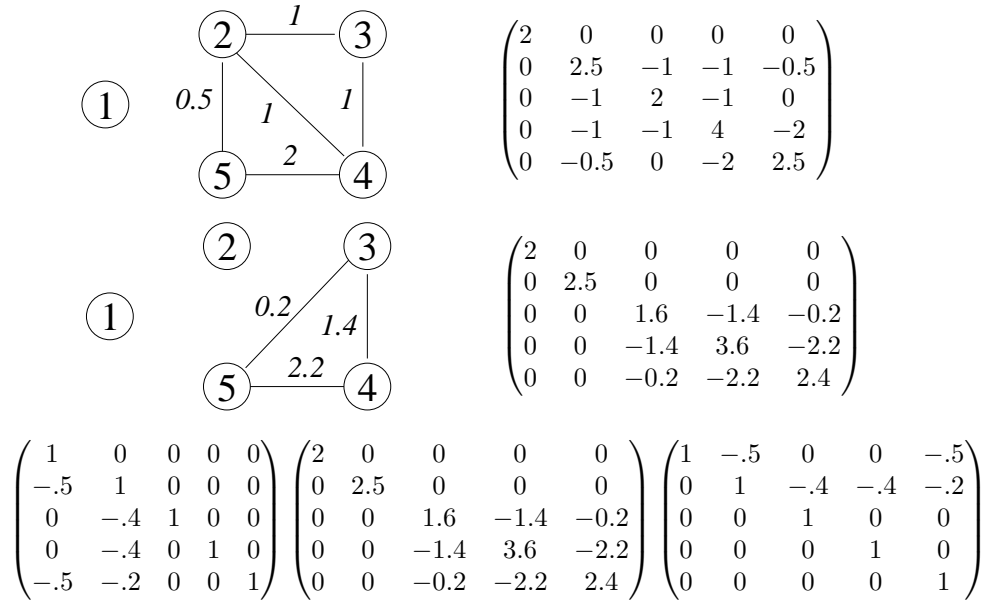


Figure 2. The first line depicts the result of eliminating vertex 1 from the graph in Figure 1. The second line depicts the result of also eliminating vertex 2. The third line presents the factorization of the Laplacian produced so far.

For example, if G is a path graph then its Laplacian will be tri-diagonal. A vertex at the end of the path has degree 1, and its elimination results in a path that is shorter by one. Thus, one may produce a Cholesky factorization of a path graph with at most $2n$ non-zero entries in time $O(n)$. One may do the same if G is a tree: a tree always has a vertex of degree 1, and its elimination results in a smaller tree. Even when dealing with a graph that is not a tree, similar ideas may be applied. Many practitioners use the Minimum Degree Ordering [TW67] or the Approximate Minimum Degree Ordering [ADD96] in an attempt to minimize the number of non-zero entries in L and the time required to compute it.

For graphs that can be disconnected into pieces of approximately the same size without removing too many vertices, one can find orderings that result in lower-triangular factors that are sparse. For example, George's Nested Dissection Algorithm [Geo73] can take as input the Laplacian of a weighted \sqrt{n} -by- \sqrt{n} grid graph and output a lower-triangular factorization with $O(n \log n)$ non-zero entries in time $O(n^{3/2})$. This algorithm was generalized by Lipton, Rose and Tarjan to apply to any planar graph [LRT79]. They also proved the more general result

that if G is a graph such that all subgraphs of G having k vertices can be divided into pieces of size at most αk (for some constant $\alpha < 1$) by the removal of at most $O(k^\sigma)$ vertices (for $\sigma > 1/2$), then the Laplacian of G has a lower-triangular factorization with at most $O(n^{2\sigma})$ non-zero entries that can be computed in time $O(n^{3\sigma})$. For example, this would hold if k -vertex subgraph of G has isoperimetric number at most $O(k^\sigma)$.

Of course, one can also use Fast Matrix Inversion [CW82] to compute the inverse of a matrix in time approximately $O(n^{2.376})$. This approach can also be used to accelerate the computation of LL^T factorizations in the algorithms of Lipton, Rose and Tarjan (see [LRT79] and [BH74]).

3.2. Iterative Methods. Iterative algorithms for solving systems of linear equations produce successively better approximate solutions. The most fundamental of these is the Conjugate Gradient algorithm. Assume for now that we wish to solve the linear system

$$A\mathbf{x} = \mathbf{b},$$

where A is a symmetric positive definite matrix. In each iteration, the Conjugate Gradient multiplies a vector by A . The number of iterations taken by the algorithm may be bounded in terms of the eigenvalues of A . In particular, the Conjugate Gradient is guaranteed to produce an ϵ -approximate solution $\tilde{\mathbf{x}}$ in at most $O(\sqrt{\kappa_f(A)} \log(1/\epsilon))$ iterations, where we say that $\tilde{\mathbf{x}}$ is an ϵ -approximate solution if

$$\|\tilde{\mathbf{x}} - \mathbf{x}\|_A \leq \epsilon \|\mathbf{x}\|_A,$$

where \mathbf{x} is the actual solution and

$$\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^T A \mathbf{x}}.$$

The Conjugate Gradient algorithm thereby reduces the problem of solving a linear system in A to the application of many multiplications by A . This can produce a significant speed improvement when A is sparse.

One can show that the Conjugate Gradient algorithm will never require more than n iterations to compute the exact solution (if one uses with exact arithmetic). Thus, if A has m non-zero entries, the Conjugate Gradient will produce solutions to systems in A in time at most $O(mn)$. In contrast, Lipton, Rose and Tarjan [LRT79] prove that if A is the Laplacian matrix of a good expander having $m = O(n)$ edges, then under every ordering the lower-triangular factor of the Laplacian has almost $n^2/2$ non-zero entries and requires almost $n^3/6$ operations to compute by the Cholesky factorization algorithm.

While Laplacian matrices are always singular, one can apply the Conjugate Gradient to solve linear equations in these matrices with only slight modification. In this case, we insist that \mathbf{b} be in the range of the matrix. This is easy to check as the null space of the Laplacian of a connected graph is spanned by the all-ones vector. The same bounds on the running time of the Conjugate Gradient then apply. Thus, when discussing Laplacian matrices we will use the *finite condition*

number κ_f , which measures the largest eigenvalue divided by the smallest non-zero eigenvalue.

Recall that expander graphs have low condition number, and so linear equations in the Laplacians of expanders can be solved quickly by the Conjugate Gradient. On the other hand, if a graph and all of its subgraphs have cuts of small isoperimetric number, then one can apply Generalized Nested Dissection [LRT79] to solve linear equations in its Laplacian quickly. Intuitively, this tells us that it should be possible to solve every Laplacian system quickly as it seems that either the Conjugate Gradient or Cholesky factorization with the appropriate ordering should be fast. While this argument cannot be made rigorous, it does inform our design of a fast algorithm.

3.3. Preconditioned Iterative Methods. Iterative methods can be greatly accelerated through the use of preconditioning. A good preconditioner for a matrix A is another matrix B that approximates A and such that it is easy to solve systems of linear equations in B . A preconditioned iterative solver uses solutions to linear equations in B to obtain accurate solutions to linear equations in A .

For example, in each iteration the Preconditioned Conjugate Gradient (PCG) solves a system of linear equations in B and multiplies a vector by A . The number of iterations the algorithm needs to find an ϵ -accurate solution to a system in A may be bounded in terms of the *relative condition number* of A with respect to B , written $\kappa(A, B)$. For symmetric positive definite matrices A and B , this may be defined as the ratio of the largest to the smallest eigenvalue of AB^{-1} . One can show that PCG will find an ϵ -accurate solution in at most $O(\sqrt{\kappa(A, B)} \log \epsilon^{-1})$ iterations. Tighter bounds can sometimes be proved if one knows more about the eigenvalues of AB^{-1} . Preconditioners have proved incredibly useful in practice. For Laplacians, incomplete Cholesky factorization preconditioners [MV77] and Multigrid preconditioners [BHM01] have proved particularly useful.

The same analysis of the PCG applies when A and B are Laplacian matrices of connected graphs, but with $\kappa_f(A, B)$ measuring the ratio of the largest to smallest non-zero eigenvalue of AB^+ , where B^+ is the Moore-Penrose pseudoinverse of B . We recall that for a symmetric matrix B with spectral decomposition

$$B = \sum_i \lambda_i \mathbf{v}_i \mathbf{v}_i^T,$$

the pseudoinverse of B is given by

$$B^+ = \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} \mathbf{v}_i \mathbf{v}_i^T.$$

That is, B projects a vector onto the image of A and then acts as the inverse of A on its image. When A and B are the Laplacian matrices of graphs, we will view $\kappa_f(A, B)$ as a measure of how well those graphs approximate one another.

4. Approximation by Sparse Graphs

Sparsification is the process of approximating a given graph G by a sparse graph H . We will say that H is an α -approximation of G if

$$\kappa_f(L_G, L_H) \leq 1 + \alpha, \quad (1)$$

where L_G and L_H are the Laplacian matrices of G and H . This tells us that G and H are similar in many ways. In particular, they have similar eigenvalues and the effective resistances in G and H between every pair of nodes is approximately the same.

The most obvious way that sparsification can accelerate the solution of linear equations is by replacing the problem of solving systems in dense matrices by the problem of solving systems in sparse matrices. Recall that the Conjugate Gradient, used as a direct solver, can solve systems in n -dimensional matrices with m non-zero entries in time $O(mn)$. So, if we could find a graph H with $O(n)$ non-zero entries that was even a 1-approximation of G , then we could quickly solve systems in L_G by using the Preconditioned Conjugate Gradient with L_H as the preconditioner, and solving the systems in L_H by the Conjugate Gradient. Each solve in H would then take time $O(n^2)$, and the number of iterations of the PCG required to get an ϵ -accurate solution would be $O(\log \epsilon^{-1})$. So, the total complexity would be

$$O((m + n^2) \log \epsilon^{-1}).$$

Sparsifiers are also employed in the fastest algorithms for solving linear equations in Laplacians, as we will later see in Section 7.

But, why should we believe that such good sparsifiers should exist? We believed it because Benczur and Karger [BK96] developed something very similar in their design of fast algorithms for the minimum cut problem. Benczur and Karger proved that for every graph G there exists a graph H with $O(n \log n / \alpha^2)$ edges such that the weight of every cut in H is approximately the same as in G . This could either be expressed by writing

$$w(\delta_H(S)) \leq w(\delta_G(S)) \leq (1 + \alpha)w(\delta_H(S)), \quad \text{for every } S \subset V,$$

or by

$$\chi_S^T L_H \chi_S \leq \chi_S^T L_G \chi_S \leq (1 + \alpha) \chi_S^T L_H \chi_S, \quad \text{for every } \chi_S \in \{0, 1\}^V. \quad (2)$$

A sparsifier H satisfies (1) if it satisfies (2) for all vectors in \mathbb{R}^V , rather than just $\{0, 1\}^V$. To distinguish Benczur and Karger's type of sparsifiers from those we require, we call their sparsifiers *cut sparsifiers* and ours *spectral sparsifiers*.

Benczur and Karger proved their sparsification theorem by demonstrating that if one forms H at random by choosing each edge of G with an appropriate probability, and re-scales the weights of the chosen edges, then the resulting graph probably satisfies their requirements. Spielman and Srivastava [SS10a] prove that a different choice probabilities results in spectral sparsifiers that also have $O(n \log n / \alpha^2)$ edges and are α -approximations of the original graph. The probability distribution turns

out to be very natural: one chooses each edge with probability proportional to the product of its weight with the effective resistance between its endpoints. After some linear algebra, their theorem follows from the following result of Rudelson and Vershynin [RV07] that lies at the intersection of functional analysis with random matrix theory.

Lemma 4.1. *Let $\mathbf{y} \in \mathbb{R}^n$ be a random vector for which $\|\mathbf{y}\| \leq M$ and*

$$\mathbf{E} [\mathbf{y}\mathbf{y}^T] = I.$$

Let $\mathbf{y}_1, \dots, \mathbf{y}_k$ be independent copies of \mathbf{y} . Then,

$$\mathbf{E} \left[\left\| \frac{1}{k} \sum_{i=1}^k \mathbf{y}_i \mathbf{y}_i^T - I \right\| \right] \leq C \frac{\sqrt{\log k}}{\sqrt{k}} M,$$

for some absolute constant C , provided that the right hand side is at most 1.

For computational purposes, the drawback of the algorithm of Spielman and Srivastava is that it requires knowledge of the effective resistances of all the edges in the graph. While they show that it is possible to approximately compute all of these at once in time $m \log^{O(1)} n$, this computation requires solving many linear equations in the Laplacian of the matrix to be sparsified. So, it does not help us solve linear equations quickly. We now examine two directions in which sparsification has been improved: the discovery of sparsifiers with fewer edges and the direct construction of sparsifiers in nearly-linear time.

4.1. Sparsifiers with a linear number of edges. Batson, Spielman and Srivastava [BSS09] prove that for every weighted graph G and every $\beta > 0$ there is a weighted graph H with at most $\lceil n/\beta^2 \rceil$ edges for which

$$\kappa_f(L_G, L_H) \leq \left(\frac{1+\beta}{1-\beta} \right)^2.$$

For $\beta < 1/10$, this means that H is a $1 + 5\beta$ approximation of G . Thus, every Laplacian can be well-approximated by a Laplacian with a linear number of edges. Such approximations were previously known to exist for special families of graphs. For example, Ramanujan expanders [LPS88, Mar88] are optimal sparse approximations of complete graphs.

Batson, Spielman and Srivastava [BSS09] prove this result by reducing it to the following statement about vectors in isotropic position.

Theorem 4.2. *Let v_1, \dots, v_m be vectors in \mathbb{R}^n such that*

$$\sum_i v_i v_i^T = I.$$

For every $\beta > 0$ there exist scalars $s_i \geq 0$, at most n/β^2 of which are non-zero, such that

$$\kappa \left(\sum_i s_i v_i v_i^T \right) \leq \left(\frac{1+\beta}{1-\beta} \right)^2.$$

This theorem may be viewed as an extension of Rudelson’s lemma. It does not concern random sets of vectors, but rather produces one particular set. By avoiding the use of random vectors, it is possible to produce a set of $O(n)$ vectors instead of $O(n \log n)$. On the other hand, these vectors now appear with coefficients s_i . We believe that these coefficients are unnecessary if all the vectors \mathbf{v}_i have the same norm. However, this statement may be non-trivial to prove as it would imply Weaver’s conjecture KS_2 , and thereby the Kadison-Singer conjecture [Wea04].

The proof of Theorem 4.2 is elementary. It involves choosing the coefficient of one vector at a time. Potential functions are introduced to ensure that progress is being made. Success is guaranteed by proving that at every step there is a vector whose coefficient can be made non-zero without increasing the potential functions. The technique introduced in this argument has also been used [SS10b] to derive an elementary proof of Bourgain and Tzafriri’s restricted invertibility principle [BT87].

4.2. Nearly-linear time computation. Spielman and Teng [ST08b] present an algorithm that takes time $O(m \log^{13} n)$ and produces ϵ sparsifiers with $O(n \log^{29} n / \epsilon^2)$ edges. While this algorithm takes nearly-linear time and is asymptotically faster than any algorithm taking time $O(m^c)$ for any $c > 1$, it is too slow to be practical. Still, it is the asymptotically fastest algorithm for producing sparsifiers that we know so far. The algorithm relies upon other graph theoretic algorithms that are interesting in their own right.

The key insight in the construction of [ST08b] is that if G has high conductance, then one can find a good sparsifier of G through a very simple random sampling algorithm. On the other hand, if G does not have high conductance then one can partition the vertices of G into two parts without removing too many edges. By repeatedly partitioning in this way, one can divide any dense graph into parts of high conductance while removing only a small fraction of its edges (see also [Tre05] and [KVV04]). One can then produce a sparsifier by randomly sampling edges from the components of high conductance, and by recursively sparsifying the remaining edges.

However, in order to make such an algorithm fast, one requires a way of quickly partitioning a graph into subgraphs of high conductance without removing too many edges. Unfortunately, we do not yet know how to do this.

Problem 1. *Design a nearly-linear time algorithm that partitions the vertices of a graph G into sets V_1, \dots, V_k so that the conductance of the induced graph on each set V_i is high (say $\Omega(1/\log n)$) and at most half of the edges of G have endpoints in different components.*

Instead, Spielman and Teng [ST08b] show that the result of $O(\log n)$ iterations of repeated approximate partitioning suffice for the purposes of sparsification.

This leaves the question of how to approximately partition a graph in nearly-linear time. Spielman and Teng [ST08a] found such an algorithm by designing an algorithm for the local clustering problem, which we describe further in Section 8.

Problem 2. *Design an algorithm that on input a graph G and an $\alpha \leq 1$ produces an α -approximation of G with $O(n/\alpha^2)$ edges in time $O(m \log n)$.*

5. Subgraph Preconditioners and Support Theory

The breakthrough that led to the work described in the rest of this survey was Vaidya’s idea of preconditioning Laplacian matrices of graphs by the Laplacians of subgraphs of those graphs [Vai90]. The family of preconditioners that followed have been referred to as subgraph or combinatorial preconditioners, and the tools used to analyze them are known as “support theory”.

Support theory uses combinatorial techniques to prove inequalities on the Laplacian matrices of graphs. Given positive semi-definite matrices A and B , we write

$$A \succcurlyeq B$$

if $A - B$ is positive semi-definite. This is equivalent to saying that for all $\mathbf{x} \in \mathbb{R}^V$

$$\mathbf{x}^T A \mathbf{x} \succcurlyeq \mathbf{x}^T B \mathbf{x}.$$

Boman and Hendrickson [BH03] show that if $\sigma_{A,B}$ and $\sigma_{B,A}$ are the least constants such that

$$\sigma_{A,B} A \succcurlyeq B \quad \text{and} \quad \sigma_{B,A} B \succcurlyeq A,$$

then

$$\lambda_{\max}(AB^+) = \sigma_{B,A}, \quad \lambda_{\min}(AB^+) = \sigma_{A,B}, \quad \text{and} \quad \kappa(A, B) = \sigma_{A,B} \sigma_{B,A}.$$

Such inequalities are natural for the Laplacian matrices of graphs.

Let $G = (V, E, w)$ be a graph and $H = (V, F, w)$ be a subgraph, where we have written w in both to indicate that edges that appear in both G and H should have the same weights. Let L_G and L_H denote the Laplacian matrices of these graphs. We then know that

$$\mathbf{x}^T L_G \mathbf{x} = \sum_{(u,v) \in E} w_{u,v} (\mathbf{x}(u) - \mathbf{x}(v))^2 \geq \sum_{(u,v) \in F} w_{u,v} (\mathbf{x}(u) - \mathbf{x}(v))^2 = \mathbf{x}^T L_H \mathbf{x}.$$

So, $L_G \succcurlyeq L_H$.

For example, Vaidya [Vai90] suggested preconditioning the Laplacian of graph by the Laplacian of a spanning tree. As we can use a direct method to solve linear equations in the Laplacians of trees in linear time, each iteration of the PCG with a spanning tree preconditioner would take time $O(m + n)$, where m is the number of edges in the original graph. In particular, Vaidya suggested preconditioning by the Laplacian of a maximum spanning tree. One can show that if T is a maximum spanning tree of G , then $(nm)L_T \succcurlyeq L_G$ (see [BGH⁺06] for details). While maximum spanning trees can be good preconditioners, this bound is not sufficient to prove it. From this bound, we obtain an upper bound of nm on the relative condition number, and thus a bound of $O(\sqrt{nm})$ on the number of iterations of PCG. However, we already know that PCG will not require more than n iterations. To obtain provably faster spanning tree preconditioners, we must measure their quality in a different way.

6. Low-Stretch Spanning Trees

Boman and Hendrickson [BH01] recognized that for the purpose of preconditioning, one should measure the stretch of a spanning tree. The concept of the *stretch* of a spanning tree was first introduced by Alon, Karp, Peleg and West [AKPW95] in an analysis of algorithms for the k -server problem. However, it can be cleanly defined without reference that problem.

We begin by defining the stretch for graphs in which every edge has weight 1. If T is a spanning tree of $G = (V, E)$, then for every edge $(u, v) \in E$ there is a unique path in T connecting u to v . When all the weights in T and G are 1, the *stretch of (u, v) with respect to T* , written $\text{st}_T(u, v)$, is the number of edges in that path. The *stretch of G with respect to T* is then the sum of the stretches of all the edges in G :

$$\text{st}_T(G) = \sum_{(u,v) \in E} \text{st}_T(u, v).$$

For a weighted graph $G = (V, E, w)$ and spanning tree $T = (V, F, w)$, the stretch of an edge $e \in E$ with respect to T may be defined by assigning a length to every edge equal to the reciprocal of its weight. The stretch of an edge $e \in E$ is then just the length of the path in T between its endpoints divided by the length of e :

$$\text{st}_T(e) = w_e \left(\sum_{f \in P} \frac{1}{w_f} \right),$$

where P is the set of edges in the path in T from u to v . This may also be viewed as the effective resistance between u and v in T divided by the resistance of the edge e . To see this, recall that the resistances of edges are the reciprocals of their weights and that the effective resistance of a chain of resistors is the sum of their resistances.

Using results from [BH03], Boman and Hendrickson [BH01] proved that

$$\text{st}_T(G)L_T \succcurlyeq L_G.$$

Alon et al. [AKPW95] proved the surprising result that every weighted graph G has a spanning tree T for which

$$\text{st}_T(G) \leq m2^{O(\sqrt{\log n \log \log n})} \leq m^{1+o(1)},$$

where m is the number of edges in G . They also showed how to construct such a tree in time $O(m \log n)$. Using these low-stretch spanning trees as preconditioners, one can solve a linear system in a Laplacian matrix to accuracy ϵ in time

$$O(m^{3/2+o(1)} \log \epsilon^{-1}).$$

Presently the best construction of low-stretch spanning trees is that of Abraham, Bartal and Neiman [ABN08], who employ the star-decomposition of Elkin, Emek, Spielman and Teng [EEST08] to prove the following theorem.

Theorem 6.1. *Every weighted graph G has a spanning tree T such that*

$$\text{st}_T(G) \leq O(m \log n \log \log n (\log \log \log n)^3) \leq O(m \log n (\log \log n)^2)$$

where m is the number of edges G . Moreover, one can compute such a tree in time $O(m \log n + n \log^2 n)$.

This result is almost tight: one can show that there are graphs with $2n$ edges and no cycles of length less than $c \log n$ for some $c > 0$ (see [Mar82] or [Bol98, Section III.1]). For such a graph G and every spanning tree T ,

$$\text{st}_T(G) \geq \Omega(n \log n).$$

We ask if one can achieve this lower bound.

Problem 3. *Determine whether every weighted graph G has a spanning tree T for which*

$$\text{st}_T(G) \leq O(m \log n).$$

If so, find an algorithm that computes such a T in time $O(m \log n)$.

It would be particularly exciting to prove a result of this form with small constants.

Problem 4. *Is it true that every weighted graph G on n vertices has a spanning tree T such that*

$$\kappa_f(L_G, L_T) \leq O(n)?$$

It turns out that one can say much more about low-stretch spanning trees as preconditioners. Spielman and Woo [SW09] prove that $\text{st}_T(G)$ equals the trace of $L_G L_T^+$. As the largest eigenvalue of $L_G L_T^+$ is at most the trace, the bound on the condition number of the graph with respect to a spanning tree follows immediately. This bound proves useful in two other ways: it is the foundation of the best constructions of preconditioners, and it tells us that low-stretch spanning trees are even better preconditioners than we believed.

Once we know that $\text{st}_T(G)$ equals the trace of $L_G L_T^+$, we know much more about the spectrum of $L_G L_T^+$ than just lower and upper bounds on its smallest and largest eigenvalues. We know that $L_G L_T^+$ cannot have too many large eigenvalues. In particular, we know that it has at most k eigenvalues larger than $\text{st}_T(G)/k$. Spielman and Woo [SW09] use this fact to prove that PCG actually only requires $O((\text{st}_T(G))^{1/3} \log 1/\epsilon)$ iterations. Kolla, Makarychev, Saberi and Teng [KMST09] observe that one could turn T into a much better preconditioner if one could just fix a small number of eigenvalues. We make their argument precise in the next section.

7. Ultra-Sparsifiers

Perhaps because maximum spanning trees do not yield worst-case asymptotic improvements in the time required to solve systems of linear equations, Vaidya [Vai90]

discovered ways of improving spanning tree preconditioners. He suggested augmenting a spanning tree preconditioner by adding $o(n)$ edges to it. In this way, one obtains a graph that looks mostly like a tree, but has a few more edges. We will see that it is possible to obtain much better preconditioners this way. It is intuitive that one could use this technique to find graphs with lower relative condition numbers. For example, if for every edge that one added to the tree one could “fix” one eigenvalue of $L_G L_T^+$, then by adding $n^2/\text{st}_T(G)$ edges one could produce an augmented graph with relative condition number at most $(\text{st}_T(G)/n)^2$. We call a graph with $n + o(n)$ edges that provides a good approximation of G an *ultra-sparsifier* of G .

We must now address the question of how one would solve a system of linear equations in an ultra-sparsifier. As an ultra-sparsifier mostly looks like a tree, it must have many vertices of degree 1 and 2. Naturally, we use Cholesky factorization to eliminate all such nodes. In fact, we continue eliminating until no vertex of degree 1 or 2 remains. One can show that if the ultra-sparsifier has $n + t$ edges, then the resulting graph has at most $3t$ edges and vertices [ST09, Proposition 4.1]. If t is sufficiently small, we could solve this system directly either by Cholesky factorization or the Conjugate Gradient. As the matrix obtained after the elimination is still a Laplacian, we would do even better to solve that system recursively. This approach was first taken by Joshi [Jos97] and Reif [Rei98]. Spielman and Teng [ST09, Theorem 5.5] prove that if one can find ultra-sparsifiers of every n -vertex graph with relative condition number $c\chi^2$ and at most $n + n/\chi$ edges, for some small constant c , then this recursive algorithm will solve Laplacian linear systems in time

$$O(m\chi \log 1/\epsilon).$$

Kolla et al. [KMST09] have recently shown that such ultrasparsifiers can be obtained from low-stretch spanning trees with

$$\chi = O(\text{st}_T(G)/n).$$

For graphs with $O(n)$ edges, this yields a Laplacian linear-equation solver with complexity

$$O(n \log n (\log \log n)^2 \log 1/\epsilon).$$

While the procedure of Kolla et al. for actually constructing the ultrasparsifiers is not nearly as fast, their result is the first to tell us that such good preconditioners exist. The next challenge is to construct them quickly.

The intuition behind the Kolla et al. construction of ultrasparsifiers is basically that explained in the first paragraph of this section. But, they cannot fix each eigenvalue of the low-stretch spanning tree by the addition of one edge. Rather, they must add a small constant number of edges to fix each eigenvalue. Their algorithm successively chooses edges to add to the low-stretch spanning tree. At each iteration, it makes sure that the edge it adds has the desired impact on the eigenvalues. Progress is measured by a refinement of the barrier function approach used by Batson, Spielman and Srivastava [BSS09] for constructing graph sparsifiers.

Spielman and Teng [ST09] obtained nearly-linear time constructions of ultra-sparsifiers by combining low-stretch spanning trees with nearly-linear time constructions of graph sparsifiers [ST08b]. They showed that in time $O(m \log^{c_1} n)$ one can produce graphs with $n + (m/k) \log^{c_2} n$ edges that k -approximate a given graph G having m edges, for some constants c_1 and c_2 . This construction of ultra-sparsifiers yielded the first nearly-linear time algorithm for solving systems of linear equations in Laplacian matrices. This has led to a search for even faster algorithms.

Two days before the day on which I submitted this paper, I was sent a paper by Koutis, Miller and Peng [KMP10] that makes tremendous progress on this problem. By exploiting low-stretch spanning trees and Spielman and Srivastava's construction of sparsifiers, they produce ultra-sparsifiers that lead to an algorithm for solving linear systems in Laplacians that takes time

$$O(m \log^2 n (\log \log n)^2 \log \epsilon^{-1}).$$

This is much faster than any algorithm known to date.

Problem 5. *Can one design an algorithm for solving linear equations in Laplacian matrices that runs in time $O(m \log n \log \epsilon^{-1})$ or even in time $O(m \log \epsilon^{-1})$?*

We remark that Koutis and Miller [KM07] have designed algorithms for solving linear equations in the Laplacians of planar graphs that run in time $O(m \log \epsilon^{-1})$.

8. Local Clustering

The problem of local graph clustering may be motivated by the following problem. Imagine that one has a massive graph, and is interesting in finding a cluster of vertices near a particular vertex of interest. Here we will define a cluster to be a set of vertices of low conductance. We would like to do this without examining too many vertices of the graph. In particular, we would like to find such a small cluster while only examining a number of vertices proportional to the size of the cluster, if it exists.

Spielman and Teng [ST08a] introduced this problem for the purpose of designing fast graph partitioning algorithms. Their algorithm does not solve this problem for every choice of initial vertex. Rather, assuming that G has a set of vertices S of low conductance, they presented an algorithm that works when started from a random vertex v of S . It essentially does this by approximating the distribution of a random walk starting at v . Their analysis exploited an extension of the connection between the mixing rate of random walks and conductance established by Lovász and Simonovits [LS93]. Their algorithm and analysis was improved by Andersen, Chung and Lang [ACL06], who used approximations of the Personal PageRank vector instead of random walks and also analyzed these using the technique of Lovász and Simonovits [LS93].

So far, the best algorithm for this problem is that of Andersen and Peres [AP09]. It is based upon the volume-biased evolving set process [MP03]. Their algorithm

satisfies the following guarantee. If it is started from a random vertex in a set of conductance ϕ , it will output a set of conductance at most $O(\phi^{1/2} \log^{1/2} n)$. Moreover, the running time of their algorithm is at most $O(\phi^{-1/2} \log^{O(1)} n)$ times the number of vertices in the set their algorithm outputs.

References

- [ABN08] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 781–790, Oct. 2008.
- [ACF⁺04] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Rcke. Optimal oblivious routing in polynomial time. *Journal of Computer and System Sciences*, 69(3):383 – 394, 2004. Special Issue on STOC 2003.
- [ACL06] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society.
- [ADD96] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.
- [AKPW95] 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(1):78–100, February 1995.
- [Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.
- [AM85] Noga Alon and V. D. Milman. λ_1 , isoperimetric inequalities for graphs, and superconcentrators. *J. Comb. Theory, Ser. B*, 38(1):73–88, 1985.
- [AM07] Dimitris Achlioptas and Frank Mcsherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2):9, 2007.
- [AP09] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 235–244, New York, NY, USA, 2009. ACM.
- [ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):1–37, 2009.
- [Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 184. IEEE Computer Society, 1996.
- [Bar98] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 161–168, 1998.
- [BGH⁺06] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006.

- [BH74] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.
- [BH01] Erik Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.
- [BH03] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3):694–717, 2003.
- [BHM01] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial, 2nd Edition*. SIAM, 2001.
- [BHV08] Erik G. Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008.
- [BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $O(n^2)$ time. In *Proceedings of The Twenty-Eighth Annual ACM Symposium On The Theory Of Computing (STOC '96)*, pages 47–55, May 1996.
- [Bol98] Béla Bollobás. *Modern graph theory*. Springer-Verlag, New York, 1998.
- [BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of computing*, pages 255–262, 2009.
- [BT87] J. Bourgain and L. Tzafriri. Invertibility of "large" sumatriicies with applications to the geometry of banach spaces and harmonic analysis. *Israel Journal of Mathematics*, 57:137–224, 1987.
- [Che70] J. Cheeger. A lower bound for smallest eigenvalue of the Laplacian. In *Problems in Analysis*, pages 195–199, Princeton University Press, 1970.
- [Chu97] Fan R. K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.
- [CW82] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*, 11(3):472–492, August 1982.
- [Dod84] Jozef Dodziuk. Difference equations, isoperimetric inequality and transience of certain random walks. *Transactions of the American Mathematical Society*, 284(2):787–794, 1984.
- [DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460, 2008.
- [EEST08] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 32(2):608–628, 2008.
- [FG07] A. Frangioni and C. Gentile. Prim-based support-graph preconditioners for min-cost flow problems. *Computational Optimization and Applications*, 36(2):271–287, 2007.
- [Fie73] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.
- [FK99] Alan Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

- [FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485 – 497, 2004. Special Issue on STOC 2003.
- [Geo73] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.
- [GL81] J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.
- [HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.
- [Jos97] Anil Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, UIUC, 1997.
- [Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.
- [KM07] Ioannis Koutis and Gary L. Miller. A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1002–1011, 2007.
- [KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd systems. March 2010. Available at <http://arxiv.org/abs/1003.2958v1>.
- [KMST09] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shanghua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. *CoRR*, abs/0912.1623, 2009.
- [KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.
- [LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- [LR99] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.
- [LRT79] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, April 1979.
- [LS88] Gregory F. Lawler and Alan D. Sokal. Bounds on the l^2 spectrum for Markov chains and Markov processes: A generalization of Cheeger’s inequality. *Transactions of the American Mathematical Society*, 309(2):557–580, 1988.
- [LS93] Lovasz and Simonovits. Random walks in a convex body and an improved volume algorithm. *RSA: Random Structures & Algorithms*, 4:359–412, 1993.
- [Mar82] G. A. Margulis. Graphs without short cycles. *Combinatorica*, 2:71–78, 1982.
- [Mar88] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, July 1988.

- [MP03] Ben Morris and Yuval Peres. Evolving sets and mixing. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 279–286, New York, NY, USA, 2003. ACM.
- [MV77] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m -matrix. *Mathematics of Computation*, 31(137):148–162, 1977.
- [Rei98] John Reif. Efficient approximate solution of sparse linear systems. *Computers and Mathematics with Applications*, 36(9):37–58, 1998.
- [RV07] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM*, 54(4):21, 2007.
- [SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, July 1989.
- [SS10a] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 2010. To appear.
- [SS10b] Daniel A. Spielman and Nikhil Srivastava. Title: An elementary proof of the restricted invertibility theorem. Available at <http://arxiv.org/abs/0911.1114>, 2010.
- [ST08a] 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. Available at <http://arxiv.org/abs/0809.3232>.
- [ST08b] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008. Available at <http://arxiv.org/abs/0808.4134>.
- [ST09] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2009. Available at <http://www.arxiv.org/abs/cs.NA/0607105>.
- [Str86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.
- [SW09] Daniel A. Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009. Available at <http://arxiv.org/abs/0903.2816>.
- [Tre05] Lucan Trevisan. Approximation algorithms for unique games. *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 197–205, Oct. 2005.
- [TW67] W.F. Tinney and J.W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801 – 1809, nov. 1967.
- [Vai90] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.

- [Var85] N. Th. Varopoulos. Isoperimetric inequalities and Markov chains. *Journal of Functional Analysis*, 63(2):215 – 239, 1985.
- [Wea04] Nik Weaver. The Kadison-Singer problem in discrepancy theory. *Discrete Mathematics*, 278(1-3):227 – 239, 2004.
- [ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proc. 20th Int. Conf. on Mach. Learn.*, 2003.

Department of Computer Science
Yale University
New Haven, CT 06520-8285
E-mail: daniel.spielman@yale.edu