SIAM J. MATRIX ANAL. APPL.
Vol. 35, No. 3, pp. 835–885

NEARLY LINEAR TIME ALGORITHMS FOR PRECONDITIONING AND SOLVING SYMMETRIC, DIAGONALLY DOMINANT LINEAR SYSTEMS*

DANIEL A. SPIELMAN† AND SHANG-HUA TENG‡

Abstract. We present a randomized algorithm that on input a symmetric, weakly diagonally dominant n-by-n matrix A with m nonzero entries and an n-vector $\boldsymbol{b}$ produces an $\tilde{\boldsymbol{x}}$ such that $\|\tilde{\boldsymbol{x}} - A^\dagger \boldsymbol{b}\|_A \leq \epsilon \|A^\dagger \boldsymbol{b}\|_A$ in expected time $O(m \log^c n \log(1/\epsilon))$ for some constant c. By applying this algorithm inside the inverse power method, we compute approximate Fiedler vectors in a similar amount of time. The algorithm applies subgraph preconditioners in a recursive fashion. These preconditioners improve upon the subgraph preconditioners first introduced by Vaidya in 1990. For any symmetric, weakly diagonally dominant matrix A with nonpositive off-diagonal entries and $k \geq 1$, we construct in time $O(m \log^c n)$ a preconditioner B of A with at most $2(n-1) + O((m/k) \log^{39} n)$ nonzero off-diagonal entries such that the finite generalized condition number $\kappa_f(A, B)$ is at most k, for some other constant c. In the special case when the nonzero structure of the matrix is planar the corresponding linear system solver runs in expected time $O(n \log^2 n + n \log n \ \log\log n \ \log(1/\epsilon))$. We hope that our introduction of algorithms of low asymptotic complexity will lead to the development of algorithms that are also fast in practice.

Key words. support theory, preconditioning, linear equation solvers, symmetric, diagonally dominant matrices, Laplacians

AMS subject classifications. 65F08, 68Q25

DOI. 10.1137/090771430

1. Introduction.

We design an algorithm with nearly optimal asymptotic complexity for solving linear systems in symmetric, weakly diagonally dominant (SDD_0) matrices. The algorithm applies a classical iterative solver, such as the preconditioned conjugate gradient or the preconditioned Chebyshev method, with a novel preconditioner that we construct and analyze using techniques from graph theory. Linear systems in these preconditioners may be reduced to systems of smaller size in linear time by use of a direct method. The smaller linear systems are solved recursively. The resulting algorithm solves linear systems in SDD_0 matrices in time that is asymptotically almost linear in their number of nonzero entries. Our analysis does not make any assumptions about the nonzero structure of the matrix and thus may be applied to the solution of the systems in SDD_0 matrices that arise in any application, such as the

*Received by the editors October 6, 2009; accepted for publication (in revised form) by B. Hendrickson January 15, 2014; published electronically July 8, 2014. This work was supported by the National Science Foundation under grants 0325630, 0324914, 0634957, 0635102, 0707522, 0964481, 1111257, and 1111270. 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.

http://www.siam.org/journals/simax/35-3/77143.html

†Department of Computer Science, Program in Applied Mathematics, Yale University, New Haven, CT 06520-8285 (spielman@cs.yale.edu).

‡Department of Computer Science, Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90089 (shanghua@usc.edu).This paper is the last in a sequence of three papers expanding on material that appeared first under the title *Nearly-Linear Time Algorithms for Graph Partitioning, Graph Sparsification, and Solving Linear Systems* [53]. The second paper, *Spectral Sparsification of Graphs* [54], contains algorithms for constructing sparsifiers of graphs, which we use in this paper to build preconditioners. The first paper, *A Local Clustering Algorithm for Massive Graphs and Its Application to Nearly Linear Time Graph Partitioning* [52], contains graph partitioning algorithms that are used to construct sparsifiers in the second paper.

$$\begin{bmatrix} 1.5 & -1.5 & 0 & 0 \\ -1.5 & 4 & -2 & -0.5 \\ 0 & -2 & 3 & -1 \\ 0 & -0.5 & -1 & 1.5 \end{bmatrix}$$

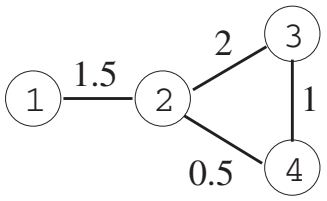

FIG. 1. *A Laplacian matrix and its corresponding weighted graph.*

solution of elliptic partial differential equations by the finite element method [56, 14], the solution of maximum flow problems [25, 20, 17], or the solution of learning problems on graphs [8, 62, 63]. More applications of the SDD_0 solver are discussed in [57]. Here, in section 7, we present a simple example of using this solver in a nearly linear time algorithm for computing an approximate Fielder vector of any weighted graph.

Graph theory drives the construction of our preconditioners. Our algorithm is best understood by first examining its behavior on Laplacian matrices—symmetric matrices with nonpositive off-diagonals and zero row sums. Each n-by-n Laplacian matrix A may be associated with a weighted graph, in which the weight of the edge between distinct vertices i and j is $-A_{i,j}$ (see Figure 1). We precondition the Laplacian matrix A of a graph G by the Laplacian matrix B of a subgraph H of G that resembles a spanning tree of G plus a few edges. The subgraph H is called an *ultra-sparsifier* of G, and its corresponding Laplacian matrix is a very good preconditioner for A: The finite generalized condition number $\kappa_f(A, B)$ is $\log^{O(1)} n$. Moreover, it is easy to solve linear equations in B. As the graph H resembles a tree plus a few edges, we may use partial Cholesky factorization to eliminate most of the rows and columns of B while incurring only a linear amount of fill. We then solve the reduced system recursively.

This paper contains two principal contributions. The first, which appears in sections 4 through 6, is the analysis of a multilevel algorithm that uses ultra-sparsifiers to solve systems of equations in SDD_0 matrices. The second, which appears in sections 8 through 11, is the construction of ultra-sparsifiers for Laplacian matrices. In the remainder of the introduction we formally define ultra-sparsifiers and the sparsifiers from which they are built. In section 2, we survey the contributions upon which we build, the improvements upon our work, and other related work. We devote section 3 to recalling the basics of support theory, explaining how the problem of solving linear equations in SDD_0 matrices may be reduced to that of solving equations in positive definite SDD_0 matrices with nonnegative off-diagonal entries, and explaining how the problem of preconditioning such matrices can be reduced to that of preconditioning Laplacian matrices.

In section 4, we state the properties we require of partial Cholesky factorizations, and we present our first algorithms for solving equations in SDD_0-matrices. These algorithms directly solve equations in the preconditioners, rather than using a recursive approach, and take time roughly $O(m^{5/4} \log^c n)$, for some constant c, for general SDD_0-matrices and time $O(n^{9/8} \log^{1/2} n)$ for SDDM-matrices with planar nonzero structure. To accelerate these algorithms, we apply our preconditioners in a recursive fashion. We analyze the complexity of these recursive algorithms in section 5, obtaining our main algorithmic results. In section 6, we show that our algorithm provides accurate answers even with computations in limited precision. In section 7, we observe that these linear system solvers yield efficient algorithms

for computing approximate Fiedler vectors, when applied inside the inverse power method.

We do not attempt to optimize the exponent of $\log n$ *in the complexity of our algorithm.* Rather, we present the simplest analysis we can find of an algorithm of complexity

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

for some constant c. Recently, Koutis, Miller, and Peng [40, 41] have discovered much simpler constructions of ultra-sparsifiers which lead to algorithms achieving such a running time for every $c > 1$. Similarly, we do not prove tight bounds on the precision required for our algorithms to work. We merely prove that $O(\log \kappa(A) \log^c n \log \epsilon^{-1})$ bits of precision suffice. We hope that the efficient preconditioners of Koutis, Miller, and Peng [40, 41] will motivate a tighter stability analysis.

1.1. Definitions and notation. We recall that a matrix A is *weakly diagonally dominant* if $A(i,i) \geq \sum_{j \neq i} |A(i,j)|$ for all i. We define SDD_0 to be the class of symmetric, weakly diagonally dominant matrices and SDDM_0 to be the class of SDD_0-matrices with nonpositive off-diagonal entries. We let SDDM be the class of positive definite SDDM_0 matrices. The SDDM-matrices are M-matrices and in particular are Stieltjes matrices. A Laplacian matrix is an SDDM_0-matrix with zero row sums.

Throughout this paper, we define the A-norm by

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

For symmetric matrices A and B, we write

$$A \preccurlyeq B$$

if $B - A$ is positive semidefinite. We recall that if A is positive semidefinite and B is symmetric, then all eigenvalues of AB are real. For a matrix B, we let $B^\dagger$ denote the Moore–Penrose pseudoinverse of B—that is, the matrix with the same nullspace as B that acts as the inverse of B on its image. We let $\kappa(A)$ denote the ratio of the largest to the smallest singular value of A. These are the largest and smallest eigenvalues when A is symmetric and positive definite.

We denote the logarithm base 2 of x by $\log x$ and the natural logarithm of x by $\ln x$.

1.2. Ultra-sparsifiers.

Definition 1.1 (ultra-sparsifiers). *A* (k,h)*-ultra-sparsifier of an* n*-by-*n SDDM*-matrix* A *with* $2m$ *nonzero off-diagonal entries is a* SDDM*-matrix* A_s *such that*

(a) $A_s \preccurlyeq A \preccurlyeq k \cdot A_s$,

(b) A_s *has at most* $2(n-1) + 2hm/k$ *nonzero off-diagonal entries,*

(c) *the set of nonzero entries of* A_s *is a subset of the set of nonzero entries of* A.

In section 11, we present a randomized algorithm that runs in expected time $O(m \log^c n)$ and that takes as input a Laplacian matrix A and a $k \geq 1$ and produces a (k,h)-ultra-sparsifier of A with probability at least $1 - 1/2n$ for

$$h = c_3 \log_2^{c_4} n, \tag{1.1}$$

where c, c_3, and c_4 are some absolute constants. As we will use these ultra-sparsifiers throughout the paper, we will define a *k-ultra-sparsifier* to be a (k,h)-ultra-sparsifier, where h satisfies (1.1).

For matrices whose graphs are planar, we present a simpler construction of (k,h)-ultra-sparsifiers with $h = O(\log n \ \log\log n)$. This simple construction exploits low-stretch spanning trees [3, 23, 1, 2] and is presented in section 10. Our construction of ultra-sparsifiers in section 11 builds upon the simpler construction but requires the use of *spectral sparsifiers* [54]. The following definition of sparsifiers will suffice for the purposes of this paper.

DEFINITION 1.2 (spectral sparsifiers). *A d-sparsifier of an n-by-n* SDDM*-matrix A is a* SDDM*-matrix A_s such that*

(a) $A_s \preccurlyeq A \preccurlyeq (5/4)A_s$,
(b) A_s *has at most dn nonzero off-diagonal entries,*
(c) *the set of nonzero entries of A_s is a subset of the set of nonzero entries of A,*
(d) *for all i,*

$$\sum_{j \neq i} \frac{A_s(i,j)}{A(i,j)} \leq 2\left|\{j : A(i,j) \neq 0\}\right|.$$

In a companion paper [54], we present a randomized algorithm `Sparsify2` that produces sparsifiers of Laplacian matrices in expected nearly linear time. As explained in section 3, this construction can trivially be extended to all SDDM-matrices.

THEOREM 1.3 (spectral sparsification). *On input an $n \times n$ Laplacian matrix A with $2m$ nonzero off-diagonal entries and a $p > 0$,* `Sparsify2` *runs in expected time $O(m \log(1/p) \log^{17} n)$ and with probability at least $1-p$ produces a $c_1 \log^{c_2}(n/p)$-sparsifier of A, for $c_2 = 34$ and some absolute constant $c_1 > 1$.*

Spielman and Srivastava [51] construct sparsifiers with $c_2 = 1$, but their construction requires the solution of linear equations in Laplacian matrices and so cannot be used to help speed up the algorithms in this paper. Their algorithm can be made reasonably fast by using the linear systems solvers of Koutis, Miller, and Peng [41]. Batson, Spielman, and Srivastava [6] have proved that there exist sparsifiers that satisfy conditions (a) through (c) of Definition 1.2 with $c_2 = 0$.

2. Related work. In this section, we explain how our results relate to other rigorous asymptotic analyses of algorithms for solving systems of linear equations. For the most part, we restrict our attention to algorithms that make structural assumptions about their input matrices, rather than assumptions about the origins of those matrices.

Throughout our discussion, we consider an n-by-n matrix with m nonzero entries. When m is large relative to n and the matrix is arbitrary, the fastest algorithms for solving linear equations are those based on fast matrix multiplication [18, 50, 60], which take time approximately $O(n^{2.37})$. The fastest algorithm for solving general sparse positive semidefinite linear systems is the conjugate gradient. Assuming computation with infinite precision, one can show that it obtains the correct answer after $O(mn)$ operations (see [58, Theorem 38.3]). To the best of our knowledge, every faster algorithm requires additional properties of the input matrix.

2.1. Special nonzero structure. In the design and analysis of direct solvers, it is standard to represent the nonzero structure of a matrix A by an unweighted graph G_A that has an edge between vertices $i \neq j$ if and only if $A_{i,j}$ is nonzero (see [22]). If this graph has special structure, there may be elimination orderings that accelerate direct solvers. If A is tridiagonal, in which case G_A is a path, then a linear system in A can be solved in time $O(n)$. Similarly, when G_A is a tree a linear system in A can be solved in time $O(n)$ (see [22]).

If the graph of nonzero entries G_A is planar, one can use generalized nested dissection [26, 43, 27] to find an elimination ordering under which Cholesky factorization can be performed in time $O(n^{1.5})$ and produces factors with at most $O(n \log n)$ nonzero entries. We will exploit these results in our algorithms for solving planar linear systems in section 4. We recall that a planar graph on n vertices has at most $3n - 6$ edges (see [32, Corollary 11.1 (c)]), so $m \leq 6n$.

2.2. Subgraph preconditioners. Our work builds on a remarkable approach to solving linear systems in Laplacian matrices introduced by Vaidya [59]. Vaidya demonstrated that a good preconditioner for a Laplacian matrix A can be found in the Laplacian matrix B of a subgraph of the graph corresponding to A. He then showed that one could bound the condition number of the preconditioned system by bounding the dilation and congestion of an embedding of the graph of A into the graph of B. By using preconditioners obtained by adding edges to maximum spanning trees, Vaidya developed an algorithm that finds ϵ-approximate solutions to linear systems in SDDM_0-matrices with at most d nonzero entries per row in time $O((dn)^{1.75} \log(1/\epsilon))$. For matrices whose corresponding graphs have special structure, such as having a bounded genus or avoiding certain minors, he obtained even faster algorithms. For example, his algorithm for solving planar systems runs in time $O((dn)^{1.2} \log(1/\epsilon))$.

As Vaidya's paper was never published and his manuscript lacked many proofs, the task of formally working out his results fell to others. Much of its content appears in the thesis of his student, Anil Joshi [35]. Chen and Toledo [16] present an experimental study of Vaidya's preconditioners, and a complete exposition of Vaidya's work along with many extensions was presented by Bern et al. [9]. Gremban, Miller, and Zagha [30, 31] explain parts of Vaidya's paper as well as extend Vaidya's techniques. Among other results, they find ways of constructing preconditioners by *adding* vertices to the graphs. Maggs et al. [44] prove that this technique may be used to construct excellent preconditioners, but it is still not clear if they can be constructed efficiently.

Gremban [30, Lemma 7.3] (see also Appendix A) presents a reduction from the problem of solving linear systems in SDD_0 matrices to that of solving linear systems in SDDM_0 matrices that are twice as large. The machinery needed to apply Vaidya's techniques directly to matrices with positive off-diagonal elements is developed in [11]. An algebraic extension of Vaidya's techniques for bounding the condition number was presented by Boman and Hendrickson [13] and later used by them [10] to prove that the low-stretch spanning trees constructed by Alon et al. [3] yield preconditioners for which the preconditioned system has condition number at most $m2^{O\left(\sqrt{\log n \log\log n}\right)}$. They thereby obtained a solver for SDDM_0 linear systems that produces ϵ-approximate solutions in time $m^{1.5+o(1)} \log(1/\epsilon)$. Through improvements in the construction of low-stretch spanning trees [23, 1, 2] and a careful analysis of the eigenvalue distribution of the preconditioned system, Spielman and Woo [55] show that when the preconditioned conjugate gradient is applied with the best low-stretch spanning tree preconditioners, the resulting linear system solver takes time at most $O(mn^{1/3} \log^{1/2} n \log(1/\epsilon))$. The preconditioners in the present paper are formed by adding edges to these low-stretch spanning trees.

The recursive application of subgraph preconditioners was pioneered in the work of Joshi [35] and Reif [47]. Reif [47] showed how to recursively apply Vaidya's preconditioners to solve linear systems in SDDM_0-matrices with planar nonzero structure and at most a constant number of nonzeros per row in time $O(n^{1+\beta} \log^c(\kappa(A)/\epsilon))$, for some constant c, for every $\beta > 0$. While Joshi's analysis is numerically much cleaner,

he only analyzes preconditioners for simple model problems. Our recursive scheme uses ideas from both these works, with some simplification. Koutis and Miller [42] have developed recursive algorithms that solve linear systems in SDDM_0-matrices with planar nonzero structure in time $O(n \log(1/\epsilon))$.

Koutis, Miller, and Peng [40, 41] have recently made substantial improvements in the construction of ultra-sparsifiers that result in algorithms for solving linear equations in SDD_0 matrices in time $O(m \log n \log^2 \log n \log(1/\epsilon))$. Their construction has the added advantage of being much simpler than ours. Subsequently, Kelner et al. [38] have designed another fast and simple SDD_0 solver. Slightly better constructions of ultra-sparsifiers have been shown to exist by Kolla et al. [39], although their construction takes longer than nearly linear time.

2.3. Other families of matrices. Subgraph preconditioners have been used to solve systems of linear equations from a few other families.

Daitch and Spielman [20] have shown how to reduce the problem of solving linear equations in symmetric M_0-matrices to the problem of solving linear equations in SDDM_0-matrices, given a factorization of the M_0-matrix of width 2 [12]. These matrices, with the required factorizations, arise in the solution of the generalized maximum flow problem by interior point algorithms.

Shklarski and Toledo [49] introduce an extension of support graph preconditioners, called *fretsaw preconditioners*, which are well suited to preconditioning finite element matrices. Daitch and Spielman [19] use these preconditioners to solve linear equations in the stiffness matrices of two-dimensional truss structures in time $O(n^{5/4} \log n \log(1/\epsilon))$.

For linear equations that arise when solving elliptic partial differential equations, other techniques supply fast algorithms. For example, multigrid methods provably run in nearly linear time when applied to the solution of some of these linear systems [15], and algorithms based on $\mathcal{H}$-matrices run in nearly linear time when given a sufficiently nice discretization [7]. Boman, Hendrickson, and Vavasis [14] have shown that the problem of solving a large class of these linear systems may be reduced to that of solving diagonally dominant systems. Thus, our algorithms may be applied to the solution of these systems.

3. Background. We will use the following propositions, whose proofs are elementary.

PROPOSITION 3.1. *If A and B are positive semidefinite matrices such that for some $\alpha, \beta > 0$,*

$$\alpha A \preccurlyeq B \preccurlyeq \beta A,$$

then A and B have the same nullspace.

PROPOSITION 3.2. *If A and B are positive semidefinite matrices having the same nullspace and $\alpha > 0$, then*

$$\alpha A \preccurlyeq B$$

if and only if

$$\alpha B^{\dagger} \preccurlyeq A^{\dagger}.$$

The following proposition establishes the equivalence of two notions of preconditioning. This proposition is called the "support lemma" in [9] and [30] and is implied by Theorem 10.1 of [5].

PROPOSITION 3.3. *If A and B are symmetric positive semidefinite matrices with the same nullspace, then all eigenvalues of $AB^{\dagger}$ lie between λ_{min} and λ_{max} if and only if*

$$\lambda_{min} B \preccurlyeq A \preccurlyeq \lambda_{max} B.$$

Following Bern et al. [9], we define the *finite generalized condition number* $\kappa_f(A, B)$ of matrices A and B having the same nullspace to be the ratio of the largest to smallest nonzero eigenvalues $AB^{\dagger}$. Proposition 3.3 tells us that $\lambda_{min} B \preccurlyeq A \preccurlyeq \lambda_{max} B$ implies $\kappa_f(A, B) \leq \lambda_{max}/\lambda_{min}$. One can use $\kappa_f(A, B)$ to bound the number of iterations taken by the preconditioned conjugate gradient algorithm to solve linear systems in A when using B as a preconditioner. Given bounds on λ_{max} and λ_{min}, one can similarly bound the complexity of the preconditioned Chebyshev method.

3.1. Preconditioning. When constructing preconditioners, we will focus our attention on the problem of preconditioning Laplacian matrices. Bern et al. [9] observe that the problem of preconditioning SDDM_0-matrices is easily reduced to that of preconditioning Laplacian matrices. We recall the reduction as we will make use of it later.

Any SDDM_0-matrix A can be decomposed as $A = A_L + A_D$, where A_L is a Laplacian matrix and A_D is a diagonal matrix with nonnegative entries. Given a Laplacian matrix B_L that preconditions A_L, we use $B = B_L + A_D$ as a preconditioner for A.

PROPOSITION 3.4 (see [9, Lemma 2.5]). *Let A be a SDDM_0-matrix and let $A = A_L + A_D$, where A_L is a Laplacian matrix and A_D is a diagonal matrix with nonnegative entries. If B_L is another Laplacian matrix on the same vertex set, then*

$$\kappa_f(A, B_L + A_D) \leq \kappa_f(A_L, B_L).$$

In particular, if $A_L \preccurlyeq B_L$, then $A \preccurlyeq B_L + A_D$. Similarly, if $B_L \preccurlyeq A_L$, then $B_L + A_D \preccurlyeq A$.

So, any algorithm for constructing sparsifiers or ultra-sparsifiers for Laplacian matrices can immediately be converted into an algorithm for constructing sparsifiers or ultra-sparsifiers for SDDM_0-matrices. This is why we restrict our attention to the problem of preconditioning Laplacian matrices in sections 10 and 11.

3.2. Solving equations. In this section, we describe how one can quickly transform the general problem of solving a system of equations in a SDD_0-matrix to the problem of solving a system of equations in an irreducible SDDM-matrix.

Recall that a symmetric matrix A is *reducible* if there is a permutation matrix P for which $P^T AP$ is a block-diagonal matrix with at least two blocks. If such a permutation exists, one can find it in linear time. A matrix that is not reducible is said to be *irreducible*. The problem of solving a linear system in a reducible matrix can be reduced to the problems of solving linear systems in each of the blocks. Accordingly, we will only consider the problem of solving linear systems in irreducible matrices. It is well-known that a symmetric matrix is irreducible if and only if its corresponding graph of nonzero entries is connected. We use this fact in the special case of Laplacian

matrices, observing that the weighted graph associated with a Laplacian matrix A has the same set of edges as G_A.

PROPOSITION 3.5. *A Laplacian matrix is irreducible if and only if its corresponding weighted graph is connected.*

It is also well-known that the nullspace of the Laplacian matrix of a connected graph is the span of the all-1's vector. Combining this fact with Proposition 3.4, one can show that the only singular irreducible SDDM_0-matrices are the Laplacian matrices.

PROPOSITION 3.6. *A singular irreducible* SDDM_0*-matrix is a Laplacian matrix, and its nullspace is spanned by the all-*1*'s vector.*

We now note that by Gremban's reduction, the problem of solving an equation of the form $A\boldsymbol{x} = \boldsymbol{b}$ for an SDD_0-matrix A can be reduced to the problem of solving a system that is twice as large in an SDDM_0-matrix (see Appendix A), without any loss of approximation quality. So, for the purposes of asymptotic complexity we need only consider the problem of solving systems in SDDM_0-matrices.

While the algorithms we develop may be naturally applied to the solution of equations in both positive definite and singular SDDM_0 matrices, it is simpler to analyze the algorithms by considering just one of these cases. We find it simpler to reduce the singular, Laplacian, case to the positive definite case and then to analyze our solvers for positive definite matrices. Let A be an irreducible Laplacian matrix. As the nullspace of A is spanned by the constant vectors, the equation $A\boldsymbol{x} = \boldsymbol{b}$ will only have a solution if the sum of the entries of $\boldsymbol{b}$ is zero. In this case, the system is underdetermined and for every solution $\boldsymbol{x}$ the vector $\boldsymbol{x} - \mathbf{1}\boldsymbol{x}(1)$ is a solution as well. Thus, we may assume that $\boldsymbol{x}(1) = 0$ and seek a solution in the remaining variables. If we let A_2 be the submatrix of A containing all but its first row and column and we let $\boldsymbol{x}_2$ and $\boldsymbol{b}_2$ denote the second through the last entries of $\boldsymbol{x}$ and $\boldsymbol{b}$, then we find

$$A_2\boldsymbol{x}_2 = \boldsymbol{b}_2.$$

It is easy to see that this system is positive definite: A_2 is a diagonally dominant SDDM-matrix, and the rows corresponding to vertices that are neighbors of vertex 1 are strictly diagonally dominant. If we obtain an approximate solution to this system $\tilde{\boldsymbol{x}}_2$ and set $\tilde{\boldsymbol{x}}$ to be zero in its first coordinate and $\tilde{\boldsymbol{x}}_2$ in the rest, then

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A = \|\tilde{\boldsymbol{x}}_2 - \boldsymbol{x}_2\|_{A_2}.$$

As $\|\boldsymbol{x}\|_A = \|\boldsymbol{x}_2\|_{A_2}$, the guarantee that our solver returns an $\tilde{\boldsymbol{x}}_2$ satisfying $\|\tilde{\boldsymbol{x}}_2 - \boldsymbol{x}_2\|_{A_2} \leq \epsilon \|\boldsymbol{x}_2\|_{A_2}$ implies that $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A \leq \epsilon \|\boldsymbol{x}\|_A$. As we have assumed that A is irreducible, its nullspace is just the span of the constant vector. So, we can bring $\tilde{\boldsymbol{x}}$ close to $A^\dagger \boldsymbol{b}$ by subtracting the average entry of $\tilde{\boldsymbol{x}}$ from each of its entries. However, this is not strictly necessary as we have

$$\left\|\tilde{\boldsymbol{x}} - A^\dagger \boldsymbol{b}\right\|_A = \|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_A \leq \epsilon \|\boldsymbol{x}\|_A = \epsilon \left\|A^\dagger \boldsymbol{b}\right\|_A.$$

So, our bound on the quality of $\tilde{\boldsymbol{x}}$ as a solution to the singular system is the same as our bound on the quality of $\tilde{\boldsymbol{x}}_2$ as a solution to the positive definite system.

4. Solvers and one-level algorithms. To solve a system in an irreducible SDDM-matrix A, we will compute an ultra-sparsifier B of A and then solve the system in A using a preconditioned iterative method. At each iteration of this method, we will need to solve a system in B. We will solve a system in B by a two-step algorithm. We will first apply Cholesky factorization repeatedly to eliminate all rows

and columns with at most one or two nonzero off-diagonal entries. As we stop the Cholesky factorization before it has factored the entire matrix, we call this process a *partial Cholesky factorization*. We then apply another solver on the remaining system. In this section, we analyze the use of a direct solver. In section 5, we obtain our fastest algorithms by solving the remaining system recursively.

4.1. Partial Cholesky factorization. The application of partial Cholesky factorization to eliminate rows and columns with at most two nonzero off-diagonal entries results in a factorization of B of the form

$$B = PLCL^T P^T,$$

where C has the form

$$C = \begin{pmatrix} I_{n-n_1} & 0 \\ 0 & A_1, \end{pmatrix},$$

P is a permutation matrix, L is nonsingular and lower triangular of the form

$$L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I_{n_1}, \end{pmatrix},$$

and every row and column of A_1 has at least three nonzero off-diagonal entries.

In the following proposition we state properties of this factorization that we will exploit. Variants of this proposition are implicit in earlier work on subgraph preconditioners [59, 35, 9].

PROPOSITION 4.1 (partial Cholesky factorization). *If B is an irreducible* SDDM*-matrix then the following hold:*

(a) *A_1 is an irreducible* SDDM*-matrix.*
(b) *If the graph of nonzero entries of B is planar, then the graph of nonzero entries of A_1 is as well.*
(c) *L has at most $3n$ nonzero entries.*
(d) *If B has $2(n-1+j)$ nonzero off-diagonal entries, then A_1 has dimension at most $2j-2$ and has at most $2(3j-3)$ nonzero off-diagonal entries.*

Proof. It is routine to verify that A_1 is diagonally dominant with nonpositive off-diagonal entries and that planarity is preserved by elimination of rows and columns with 2 or 3 nonzero entries, as these correspond to vertices of degree 1 or 2 in the graph of nonzero entries. It is similarly routine to observe that these eliminations preserve irreducibility and singularity.

To bound the number of entries in L, we note that for each row and column with 1 nonzero off-diagonal entry that is eliminated, the corresponding column in L has 2 nonzero entries, and that for each row and column with 2 nonzero off-diagonal entries that is eliminated, the corresponding column in L has 3 nonzero entries.

To bound n_1, the dimension of A_1, first observe that the elimination of a row and column with 1 or 2 nonzero off-diagonal entries decreases both the dimension by 1 and the number of nonzero entries by 2. So, A_1 will have $2(n_1 - 1 + j)$ nonzero off-diagonal entries. As each row in A_1 has at least 3 nonzero off-diagonal entries, we have

$$2(n_1 - 1 + j) \geq 3n_1,$$

which implies $n_1 \leq 2j-2$. The bound on the number of nonzero off-diagonal entries in A_1 follows immediately. □

We name the algorithm that performs this factorization `PartialChol` and invoke it with the syntax

$$(P, L, A_1) = \texttt{PartialChol}(B).$$

We remark that `PartialChol` can be implemented to run in linear time.

4.2. One-level algorithms. Before analyzing the algorithm in which we solve systems in A_1 recursively, we pause to examine the complexity of an algorithm that applies a direct solver to systems in A_1. While the results in this subsection are not necessary for the main claims of our paper, we hope they will provide intuition.

If we are willing to ignore numerical issues, we may apply the conjugate gradient algorithm to directly solve systems in A_1 in $O(n_1 m_1)$ operations [58, Theorem 38.3], where m_1 is the number of nonzero entries in A_1. In the following theorem, we examine the performance of the resulting algorithm.

THEOREM 4.2 (general one-level algorithm). *Let A be an irreducible n-by-n SDDM-matrix with $2m$ nonzero off-diagonal entries. Let B be a $\sqrt{m}$-ultra-sparsifier of A. Let $(P, L, A_1) = \texttt{PartialChol}(B)$. Consider the algorithm that solves systems in A by applying PCG with B as a preconditioner and solves each system in B by performing backward substitution on its partial Cholesky factor, solving the inner system in A_1 by conjugate gradient used as an exact solver, and performing forward substitution on its partial Cholesky factor. Then for every right-hand side $\boldsymbol{b}$, after*

$$O(m^{1/4} \log(1/\epsilon))$$

iterations, comprising

$$O(m^{5/4} \log^{2c_4} n \log(1/\epsilon))$$

arithmetic operations, the algorithm will output an approximate solution $\tilde{\boldsymbol{x}}$ satisfying

$$\left\| \tilde{\boldsymbol{x}} - A^{-1}\boldsymbol{b} \right\|_A \leq \epsilon \left\| A^{-1}\boldsymbol{b} \right\|_A. \tag{4.1}$$

Proof. As $\kappa_f(A, B) \leq \sqrt{m}$, we may apply the standard analysis of PCG [5] to show that (4.1) will be satisfied after $O(m^{1/4} \log(1/\epsilon))$ iterations. To bound the number of operations in each iteration, note that B has at most $2(n-1)+O(\sqrt{m}\log^{c_4} n)$ nonzero off-diagonal entries. So, Proposition 4.1 implies m_1 and n_1 are both $O(\sqrt{m}\log^{c_4} n)$. Thus, the time required to solve each inner system in A_1 by the conjugate gradient is at most $O(m_1 n_1) = O(m \log^{2c_4} n)$. As A is irreducible, $m \geq n-1$, and so this upper bounds the number of operations that must be performed in each iteration. □

When the graph of nonzero entries of A is planar, we may precondition A by using the algorithm `UltraSimple`, presented in section 10, instead of `UltraSparsify`. As the matrix A_1 produced by applying partial Cholesky factorization to the output of `UltraSimple` is also planar, we can solve the linear systems in A_1 by the generalized nested dissection algorithm of Lipton, Rose, and Tarjan [43]. This algorithm uses graph separators to choose a good order for Cholesky factorization. The Cholesky factorization is then computed in time $O(n_1^{3/2})$. The resulting Cholesky factors only

have $O(n_1 \log n_1)$ nonzero entries, and so each linear system in A_1 may be solved in time $O(n_1 \log n_1)$, after the Cholesky factors have been computed.

THEOREM 4.3 (planar one-level algorithm). *Let A be an n-by-n planar, irreducible* SDDM*-matrix with $2m$ nonzero off-diagonal entries. Consider the algorithm that solves linear systems in A by using PCG with the preconditioner*

$$B = \texttt{UltraSimple}(A, n^{3/4} \log^{1/3} n),$$

solves systems in B by applying `PartialChol` *to factor B into*[1] $PL[I, 0; 0, A_1]L^T P^T$, *and uses generalized nested dissection to solve systems in A_1. For every right-hand side $\boldsymbol{b}$, this algorithm computes an $\tilde{\boldsymbol{x}}$ satisfying*

$$\left\| \tilde{\boldsymbol{x}} - A^\dagger \boldsymbol{b} \right\|_A \leq \epsilon \left\| A^\dagger \boldsymbol{b} \right\|_A \tag{4.2}$$

in time

$$O\big(n^{9/8} \log^{1/2} n \log(1/\epsilon)\big).$$

Proof. First, recall that the planarity of A implies $m \leq 3n$. Thus, the time taken by `UltraSimple` is dominated by the time taken by `LowStretch`, which is $O(n \log n \ \log\log n)$ (see Theorem 10.1).

By Theorems 10.1 and 10.5, the matrix B has at most $2(n-1) + 6n^{3/4} \log^{1/3} n$ nonzero off-diagonal entries and

$$\kappa_f(A, B) = O\big(n^{1/4} \log^{2/3} n \log\log n\big) \leq O\big(n^{1/4} \log n\big).$$

Again, standard analysis of PCG [5] tells us that the algorithm will require at most

$$O\big(n^{1/8} \log^{1/2} n \log(1/\epsilon)\big)$$

iterations to guarantee that (4.2) is satisfied.

By Proposition 4.1, the dimension of A_1, n_1, is at most $6n^{3/4} \log^{1/3} n$. Before beginning to solve the linear system, the algorithm will spend

$$O(n_1^{3/2}) = O((n^{3/4} \log^{1/3} n)^{3/2}) = O(n^{9/8} \log^{1/2} n)$$

time using generalized nested dissection [43] to permute and Cholesky factor the matrix A_1. As the factors obtained will have at most $O(n_1 \log n_1) \leq O(n)$ nonzeros, each iteration of the PCG will require at most $O(n)$ steps. So, the total complexity of the application of the PCG will be

$$O\big(n \cdot \big(n^{1/8} \log^{1/2} n \log(1/\epsilon)\big)\big) = O\big(n^{9/8} \log^{1/2} n \log(1/\epsilon)\big),$$

which dominates the time required by the call to `UltraSimple` and to compute the Cholesky factors. ∎

We remark that the algorithm of Lipton, Rose, and Tarjan [43] can be accelerated by the use of algorithms for fast matrix inversion [18, 50, 60]. One can similarly accelerate our planar one-level algorithm.

[1] Where $[I, 0; 0, A_1]$ denotes the diagonal block matrix defined by I and A_1 as specified in MATLAB notaton.

5. The recursive solver. In our recursive algorithm for solving linear equations, we solve linear equations in a matrix A by computing an ultra-sparsifier B, using partial Cholesky factorization to reduce it to a matrix A_1, and then solving the system in A_1 recursively. Of course, we compute all the necessary ultra-sparsifiers and Cholesky factorizations just once at the beginning of the algorithm.

In this section we assume infinite precision arithmetic. We defer an analysis of the impact of limited precision to the next section.

To specify the recursive algorithm for an n-by-n matrix, we first set the parameter

$$k = (14h + 1)^2, \tag{5.1}$$

where we recall that the parameter h is determined by the quality of the ultra-sparsifiers we can compute (see (1.1)),

We use the algorithm `BuildPreconditioners` to build the sequence of preconditioners and Cholesky factors. In section 11, we define the routine `UltraSparsify` for weighted graphs and thus implicitly for Laplacian matrices. To define `UltraSparsify` for general irreducible SDDM-matrices A, we express A as a sum of matrices A_L and A_D as explained in Proposition 3.4, and we set

$$\texttt{UltraSparsify}(A, k) = A_D + \texttt{UltraSparsify}(A_L, k).$$

In the algorithms and analyses below, we let $\dim(A)$ denote the dimension of A and let $\text{noff}(A)$ denote the number of nonzero off-diagonal entries in the upper-triangular portion of A.

`BuildPreconditioners`(A_0),

1. Set $i = 0$, $h = c_3 \log_2^{c_4} \dim(A_0)$ (as in (1.1)) and $k = (14h + 1)^2$ (as in (5.1)).
2. Repeat
   (a) $i = i + 1$.
   (b) $B_i = \texttt{UltraSparsify}(A_{i-1}, k)$.
   (c) $(P_i, L_i, A_i) = \texttt{partialChol}(B_i)$.
   Until A_i has dimension less than $66h + 6$.
3. Set $\ell = i$.
4. Compute $Z_\ell = {A_\ell}^{-1}$.

We now make a few observations about the sequence of matrices this algorithm generates.

PROPOSITION 5.1 (recursive preconditioning). *If A_0 is an irreducible* SDDM*-matrix, and for each i the matrix B_i is a k-ultra-sparsifier of A_{i-1}, then*

(a) *for $i \geq 1$, $\text{noff}(A_i) \leq (3h/k) \cdot \text{noff}(A_{i-1})$;*
(b) *for $i \geq 1$, $\dim(A_i) \leq (2h/k) \cdot \text{noff}(A_{i-1})$;*
(c) *for $i \geq 1$, $\dim(B_i) = \dim(A_{i-1})$;*
(d) *each of B_i and A_i is an irreducible* SDDM*-matrix;*
(e) $\ell \leq 2 \log_{4h} n$.

Proof. Let n_i be the dimension of A_i. Definition 1.1 tells us that

$$\text{noff}(B_i) \leq n_{i-1} - 1 + h \cdot \text{noff}(A_{i-1})/k.$$

Parts (a), (b), and (d) now follow from Proposition 4.1. Part (c) is obvious. Part (e) follows from part (a). ∎

Our recursive solver will use each matrix B_i as a preconditioner for A_{i-1}. But rather than solve systems in B_i directly, it will reduce these to systems in A_i, which will in turn be solved recursively. Our solver will use the preconditioned Chebyshev method, instead of the preconditioned conjugate gradient. This choice is dictated by the requirements of our analysis rather than by common sense. Our preconditioned Chebyshev method will not take the preconditioner B_i as input. Rather, it will take a subroutine $\texttt{solve}_{B_i}$ that produces approximate solutions to systems in B_i. So that we can guarantee that our solvers will be linear operators, we will fix the number of iterations that each will perform, as opposed to allowing them to terminate upon finding a sufficiently good solution.

For simplicity, we use the original Chebyshev iterative method [29], as presented by Axelsson [5, section 5.3]. While this variant is not numerically stable, it will not matter in this section in which we ignore numerical issues. In particular, when one assumes infinite precision, this algorithm becomes identical to its stable variants. In the next section, we will show that our use of the algorithm for a small number of iterations limits its instability.

$\boldsymbol{x} = \texttt{precondCheby}(A, \boldsymbol{b}, f(\cdot), t, \lambda_{min}, \lambda_{max})$
(0) Set $\boldsymbol{x} = \boldsymbol{0}$ and $\boldsymbol{r} = f(\boldsymbol{b})$.
(1) for $i = 1, \ldots, t$,
(a) Set $\theta_i = (2i-1)\pi/2t$ and $\tau_i = ((\cos\theta_i)(\lambda_{max} - \lambda_{min})/2 + (\lambda_{max} + \lambda_{min})/2)^{-1}$.
(b) Set $\boldsymbol{x} = \boldsymbol{x} - \tau_i f(A\boldsymbol{x}) + \tau_i \boldsymbol{r}$.

Note that the linear operator f in the input of `precondCheby` is the placeholder where we will provide an (approximate) inverse function for our preconditioner when applying the Chebyshev iterative method.

PROPOSITION 5.2 (linear Chebyshev). *Let A be a positive definite matrix and f be a positive definite, symmetric linear operator such that for some $\lambda_{max} \geq \lambda_{min} > 0$*

$$\lambda_{min} f^{-1} \preccurlyeq A \preccurlyeq \lambda_{max} f^{-1}. \tag{5.2}$$

Let $\epsilon < 1$ and let

$$t \geq \left\lceil \frac{1}{2}\sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \ln\frac{2}{\epsilon} \right\rceil. \tag{5.3}$$

Then, the function $\texttt{precondCheby}(A, \boldsymbol{b}, f, t, \lambda_{min}, \lambda_{max})$ *is a symmetric linear operator in $\boldsymbol{b}$. Moreover, if Z is the matrix realizing this operator, then*

$$(1-\epsilon)Z^{-1} \preccurlyeq A \preccurlyeq (1+\epsilon)Z^{-1}.$$

Proof. An inspection of the pseudocode reveals that the function computed by `precondCheby` can be expressed as a sum of monomials of the form $(fA)^i f$, from which it follows that this function is a symmetric linear operator.

Standard analyses of the preconditioned Chebyshev algorithm [5, section 5.3] imply that for all $\boldsymbol{b}$,

$$\left\| Z\boldsymbol{b} - A^{-1}\boldsymbol{b} \right\|_A \leq \epsilon \left\| A^{-1}\boldsymbol{b} \right\|_A.$$

Now, let λ be any eigenvalue of ZA, let $\boldsymbol{v}$ be the corresponding eigenvector, and let $\boldsymbol{b} = A\boldsymbol{v}$. We then have

$$\epsilon \|\boldsymbol{v}\|_A \geq \|ZA\boldsymbol{v} - \boldsymbol{v}\|_A = |\lambda - 1| \|\boldsymbol{v}\|_A .$$

So, $|\lambda - 1| \leq \epsilon$. Applying Proposition 3.3, we obtain

$$(1-\epsilon)Z^{-1} \preccurlyeq A \preccurlyeq (1+\epsilon)Z^{-1}. \qquad \square$$

We can now state the subroutine $\texttt{solve}_{B_i}$ for $i = 1, \ldots, \ell$.

$\boldsymbol{x} = \texttt{solve}_{B_i}(\boldsymbol{b})$

1. Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1 + 2e^{-2})k$ and $t = \lceil 1.33\sqrt{k} \rceil$, where k is as set in (5.1) and in `BuildPreconditioners` for the system A_0.
2. Set $\boldsymbol{s} = L_i^{-1} P_i^{-1} \boldsymbol{b}$.
3. Write $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{s}_0 \\ \boldsymbol{s}_1 \end{pmatrix}$, where the dimension of $\boldsymbol{s}_1$ is the size of A_i.
4. Set $\boldsymbol{y}_0 = \boldsymbol{s}_0$, and
   ((a)) if $i = \ell$, set $\boldsymbol{y}_1 = Z_\ell \boldsymbol{s}_1$
   ((b)) else, set $\boldsymbol{y}_1 = \texttt{precondCheby}(A_i, \boldsymbol{s}_1, \texttt{solve}_{B_{i+1}}, t, \lambda_{min}, \lambda_{max})$.
5. Set $\boldsymbol{x} = P_i^{-T} L_i^{-T} \begin{pmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_1 \end{pmatrix}$.

We have chosen the parameters λ_{min}, λ_{max}, and t so that inequality (5.3) holds for $\epsilon = 2e^{-2}$. Our recursive algorithm only requires the solution of the systems B_i to some small constant error. The constants given here are merely a simple choice that suffices. It might be possible to obtain constant-factor improvements in running time by the choice of better constants.

We note that we apply L_i^{-T} and L_i^{-1} by forward and backward substitution, rather than by constructing the inverses.

LEMMA 5.3 (correctness of $\texttt{solve}_{B_i}$). *If A is an irreducible* SDDM*-matrix and $B_i \preccurlyeq A_{i-1} \preccurlyeq kB_i$ for all $i \geq 1$, then for $1 \leq i \leq \ell$,*

(a) *the function $\texttt{solve}_{B_i}$ is a symmetric linear operator;*

(b) *the function $\texttt{precondCheby}(A_{i-1}, \boldsymbol{b}, \texttt{solve}_{B_i}, t, \lambda_{min}, \lambda_{max})$ is a symmetric linear operator in $\boldsymbol{b}$;*

(c) *if $i \leq \ell - 1$ and Z_i is the symmetric matrix such that*

$$Z_i \boldsymbol{s}_1 = \texttt{precondCheby}(A_i, \boldsymbol{s}_1, \texttt{solve}_{B_{i+1}}, t, \lambda_{min}, \lambda_{max}),$$

then

$$(1 - 2e^{-2}){Z_i}^{-1} \preccurlyeq A_i \preccurlyeq (1 + 2e^{-2}){Z_i}^{-1};$$

(d)

$$(1 - 2e^{-2}){\texttt{solve}_{B_i}}^{-1} \preccurlyeq B_i \preccurlyeq (1 + 2e^{-2}){\texttt{solve}_{B_i}}^{-1}.$$

Proof. We first prove (a) and (b) by reverse induction on i. The base case of our induction is when $i = \ell$, in which case `BuildPreconditioners` sets ${Z_\ell} = {A_\ell}^{-1}$, and so

$$\texttt{solve}_{B_\ell} = P_\ell^{-T} L_\ell^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_\ell \end{pmatrix} L_\ell^{-1} P_\ell^{-1},$$

which is obviously a symmetric linear operator. Given that $\texttt{solve}_{B_i}$ is a symmetric linear operator, part (b) for A_{i-1} follows from Proposition 5.2. Given that (b) holds for A_i and that the call to `precondCheby` is realized by a symmetric matrix Z_i, we then have that

$$\texttt{solve}_{B_i} = P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix} L_i^{-1} P_i^{-1}$$

is a symmetric linear operator. We may thereby establish that (a) and (b) hold for all $\ell \geq i \geq 1$.

We now prove properties (c) and (d), again by reverse induction. By construction $Z_\ell = {A_\ell}^{-1}$, so (c) holds for $i = \ell$. To see that if (c) holds for i, then (d) does also, note that

$$\begin{aligned}
(1 - 2e^{-2}){Z_i}^{-1} &\preccurlyeq A_i && \text{implies} \\
(1 - 2e^{-2}){A_i}^{-1} &\preccurlyeq Z_i && \text{by Proposition 3.2, which implies} \\
(1 - 2e^{-2}) \begin{pmatrix} I & 0 \\ 0 & {A_i}^{-1} \end{pmatrix} &\preccurlyeq \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix}, && \text{which implies}
\end{aligned}$$

$$\begin{aligned}
(1 - 2e^{-2}){B_i}^{-1} &= (1 - 2e^{-2}) P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & {A_i}^{-1} \end{pmatrix} L_i^{-1} P_i^{-1} \quad \text{(by Proposition 4.1(e))} \\
&\preccurlyeq P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix} L_i^{-1} P_i^{-1} \\
&= \texttt{solve}_{B_i},
\end{aligned}$$

which by Proposition 3.2 implies $(1 - 2e^{-2}){\texttt{solve}_{B_i}}^{-1} \preccurlyeq B_i$. The inequality $B_i \preccurlyeq (1 + 2e^{-2}){\texttt{solve}_{B_i}}^{-1}$ may be established similarly.

To show that when (d) holds for i then (c) holds for $i - 1$, note that (d) and $B_i \preccurlyeq A_{i-1} \preccurlyeq k \cdot B_i$ imply

$$(1 - 2e^{-2}){\texttt{solve}_{B_i}}^{-1} \preccurlyeq A_{i-1} \preccurlyeq k(1 + 2e^{-2}){\texttt{solve}_{B_i}}^{-1}.$$

So, (c) for $i - 1$ now follows from Proposition 5.2 and the fact that $\lambda_{min}, \lambda_{max}$, and t have been chosen so that inequality (5.3) is satisfied with $\epsilon = 2e^{-2}$. ∎

LEMMA 5.4 (complexity of $\texttt{solve}_{B_i}$). *If A_0 is a positive definite irreducible, n-by-n SDDM-matrix with $2m$ nonzero off-diagonal entries and each B_i is a k-ultra-sparsifier of A_{i-1}, then $\texttt{solve}_{B_1}$ runs in time*

$$O(n + m).$$

Proof. Let T_i denote the running time of $\texttt{solve}_{B_i}$. We will prove by reverse induction on i that there exists a constant c such that

$$T_i \leq c\left(\dim\left(B_i\right) + (\gamma h + \delta)(\text{noff}\left(A_i\right) + \dim\left(A_i\right))\right), \tag{5.4}$$

where

$$\gamma = 196 \quad \text{and} \quad \delta = 15.$$

This will prove the lemma as $\dim(B_1) = \dim(A_0) = n$, and Proposition 5.1 implies

$$(\gamma h + \delta)(\text{noff}(A_1) + \dim(A_1)) \leq (\gamma h + \delta)\frac{5hm}{k} \leq m\frac{5\gamma h^2 + 5\delta h}{(14h+1)^2} = O(m).$$

To prove (5.4), we note that there exists a constant c so that steps 2 and 5 take time at most $c(\dim(B_i))$ (by Proposition 4.1), step 4(a) takes time at most $c(\dim(A_\ell)^2)$, and step 4(b) takes time at most $t(c \cdot \dim(A_i) + c \cdot \text{noff}(A_i) + T_{i+1})$, where t is as defined on step 1 of $\texttt{solve}_{B_i}$.

The base case of our induction will be $i = \ell$, in which case the preceding analysis implies

$$\begin{aligned} T_\ell &\leq c\big(\dim(B_\ell) + \dim(A_\ell)^2\big) \\ &\leq c\left(\dim(B_\ell) + (66h+6)\dim(A_\ell)\right) \quad \text{(by step 2 of } \texttt{BuildPreconditioners}\text{)}, \end{aligned}$$

which satisfies (5.4). We now prove (5.4) is true for $i < \ell$, assuming it is true for $i+1$. We have

$$\begin{aligned} T_i &\leq c\left(\dim(B_i)\right) + t(c \cdot \dim(A_i) + c \cdot \text{noff}(A_i) + T_{i+1}) \\ &\leq c\big[\dim(B_i) + t\big(\dim(A_i) + \text{noff}(A_i) \\ &\qquad + \dim(B_{i+1}) + (\gamma h + \delta)(\text{noff}(A_{i+1}) + \dim(A_{i+1}))\big)\big] \end{aligned}$$

(by the induction hypothesis)

$$\leq c\left[\dim(B_i) + t\big(2\ \dim(A_i) + \text{noff}(A_i) + (\gamma h + \delta)(5\ \text{noff}(A_i)\, h/k)\big)\right]$$

(by Proposition 5.1)

$$\leq c\left[\dim(B_i) + t\left(2\ \dim(A_i) + 6\ \text{noff}(A_i)\right)\right],$$

as $\gamma h^2 + \delta h \leq k$. As

$$6t \leq 6 \cdot (1.33(14h+1) + 1) \leq \gamma h + \delta,$$

we have proved that (5.4) is true for i as well. ∎

We now state and analyze our ultimate solver.

$\boldsymbol{x} = \texttt{solve}(A, \boldsymbol{b}, \epsilon)$

1. Set $h = c_3 \log_2^{c_4} \dim(A)$ (as in (1.1)) and $k = (14h+1)^2$ (as in (5.1)).
   Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1 + 2e^{-2})k$ and $t = \lceil 0.67\sqrt{k}\ln(2/\epsilon)\rceil$.
2. Run $\texttt{BuildPreconditioners}(A)$.
3. $\boldsymbol{x} = \texttt{precondCheby}(A, \boldsymbol{b}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max})$.

THEOREM 5.5 (nearly linear time solver). *On input an irreducible n-by-n SDDM-matrix A with $2m$ nonzero off-diagonal entries and an n-vector $\boldsymbol{b}$, with probability at least $1 - 1/50$, $\texttt{solve}(A, \boldsymbol{b}, \epsilon)$ runs in time*

$$O(m \log^{c_4} m \log(1/\epsilon)) + m \log^c m,$$

where c is some constant and c_4 is defined in (1.1)*, and produces an $\tilde{\boldsymbol{x}}$ satisfying*

$$\left\|\tilde{\boldsymbol{x}} - A^{-1}\boldsymbol{b}\right\|_A \leq \epsilon \left\|A^{-1}\boldsymbol{b}\right\|_A.$$

Proof. By Proposition 5.1, the numbers $\mathrm{noff}(A_i)$ are geometrically decreasing, and $\ell \leq 2\log_{4h} n$. So we may use Theorem 11.5 to show that the time required to build the preconditioners is at most $m \log^{O(1)} m$. If each B_i is a k-ultra-sparsifier of A_{i-1}, then the bound on the A-norm of the output follows by an analysis similar to that used to prove Lemma 5.3. In this case, we may use Lemma 5.4 to bound on the running time of step 3 by

$$O(mt) = O(m\sqrt{k}\log(1/\epsilon)) = O\left(m \log^{c_4} n \log(1/\epsilon)\right).$$

Also by Theorem 11.5, the probability that there is some B_i that is not a k-ultra-sparsifier of A_{i-1} is at most

$$\sum_i \frac{1}{2\ \dim(B_i)} \leq \frac{\ell}{2(66h+6)} \leq \frac{2\log_{4h} n}{2(66h+6)} < 1/50,$$

assuming $c_3, c_4 \geq 1$. ∎

If the nonzero structure of A is planar, then by Theorem 10.5, we can replace all the calls to `UltraSparsify` in the above algorithm with calls to `UltraSimple`. By Theorem 10.1, this is like having (k,h)-ultra-sparsifiers with $h = O(\log n \log\log n)$. Thus, the same analysis goes through with $h = O(\log n \log\log n)$, and the resulting linear system solver runs in time

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

6. A crude stability analysis. We will now show that the recursive solver described in Theorem 5.5 works when all the computations are carried out with limited precision. In particular, we argue that $O(\log \kappa(A) \log^c n \log \epsilon^{-1})$ bits of precision are sufficient, where c is a constant independent of n and A. While this bound is rather weak by the standards of numerical linear algebra, it is sufficient for establishing many bounds on the asymptotic complexity of algorithms, such as those in [17, 20, 36, 37]. (Kelner and Madry's result [36] was substantially improved by James Propp, who will be added as a coauthor of the journal version.) We hope to one day see a better bound. The main bottleneck in our analysis is that we have been unable to find a good analysis of the stability of the preconditioned Chebyshev method.[2]

As both the condition number and smallest eigenvalue of Laplacian matrices will play a substantial role in our analysis, we briefly relate these quantities to the weights of edges in the corresponding graph. It follows from Geršgorin's circle theorem that the largest eigenvalue of an SDD_0-matrix is at most twice its largest diagonal entry. (For the special case of Laplacians, see [4].) A simple lower bound on the smallest eigenvalue of an irreducible SDDM matrix in terms of the lowest weight of an edge in the corresponding graph follows.

LEMMA 6.1. *Let G be a connected weighted graph and let A be either the Laplacian matrix of G or a principal square submatrix of the Laplacian. Then the smallest nonzero eigenvalue of A is at least* $\min(8w/n^2, w/n)$*, where w is the least weight of an edge of G and n is the dimension of A.*

Proof. Fiedler [24] proved that the smallest eigenvalue of the Laplacian of a connected, unweighted graph with n vertices is at least $2(1-\cos(\pi/n))$, which is at least $8/n^2$ for $n \geq 2$. In the weighted case, this implies that the smallest eigenvalue of the Laplacian is at least $8w/n^2$ provided that all edge weights are at least w.

[2]While Golub and Overton [28] suggest that such a stability analysis should follow from the techniques they employ, the derivation of such a result is beyond the scope of the present paper.

We now consider a submatrix of such a Laplacian. Let S be the set of vertices corresponding to the rows and columns of the submatrix. The submatrix will have one diagonal block for each connected component of S. Let S_1 be such a connected component. The submatrix induced on S_1 can be decomposed into the sum of Laplacian matrix A_1 and a diagonal matrix D_1. By the previous argument, the smallest nonzero eigenvalue of that Laplacian is at least $8w/n^2$. On the other hand, when we multiply the unit zero eigenvector of that Laplacian by D_1 we get $\mathbf{1}^T D_1 \mathbf{1}/|S_1|$. The numerator equals the sum of the weights of edges on the boundary of S_1, which is at least w. So, the smallest eigenvalue of the matrix induced on S_1 is at least $\min(8w/n^2, w/n)$. □

We begin our analysis by asserting that the algorithm `UltraSparsify` is purely combinatorial and thus very stable. It requires precision at most a polynomial in n times the ratio of the largest to the smallest nonzero off-diagonal entry of its input. This can be seen from an examination of the routines that it calls: `RootedUltraSparsify`, presented in section 11, and `Sparsify2` from [54]. In fact, the algorithm would barely suffer from rounding the weights of all edges in its input graph to powers of two.

We assume in the rest of this section that computations are performed with precision u, basing our analysis on those presented by Higham [33]. To avoid the use of the notation $O(u^2)$, we employ Higham's [33] notation

$$\gamma_j = \frac{uj}{1-uj}. \tag{6.1}$$

We first address the issue that the matrices computed by `partialChol` will not be exactly the intended matrices by observing they are close enough to provide good preconditioners.

LEMMA 6.2. *Let L_i and A_i be the matrices that would be output by* `partialChol` *on input B_i if it were run with infinite precision, and let $\widehat{L}_i$ and $\widehat{A}_i$ be the matrices that are returned when it is run with precision u. Let*

$$\widehat{B}_i = P\widehat{L}_i \begin{pmatrix} I & 0 \\ 0 & \widehat{A}_i, \end{pmatrix} \widehat{L}_i^T P^T.$$

Then,

$$(1 - n\gamma_{n+1}\kappa(B_i))\, B_i \preccurlyeq \widehat{B}_i \preccurlyeq (1 - n\gamma_{n+1}\kappa(B_i))^{-1}\, B_i.$$

Proof (sketch). Following the proof of Lemma 2.1 in [21] (see also [33, Theorem 10.5]), we can show that every entry in $B_i - \widehat{B}_i$ is at most $\gamma_{n+1} \max_j B_i(j,j)$. This implies that the norm of $B_i - \widehat{B}_i$ is at most $n\gamma_{n+1} \max_j B_i(j,j)$. As B_i is a positive semidefinite matrix, $\max_j B_i(j,j) \le \lambda_{max}(B_i)$. So, for all $\boldsymbol{x}$,

$$\left| \frac{\boldsymbol{x}^T \widehat{B}_i \boldsymbol{x} - \boldsymbol{x}^T B_i \boldsymbol{x}}{\boldsymbol{x}^T B_i \boldsymbol{x}} \right| \le \frac{n\gamma_{n+1}\lambda_{max}(B_i)}{\lambda_{min}(B_i)} = n\gamma_{n+1}\kappa(B_i).$$

The lemma follows. □

We now prove that the matrices produced by the routine `BuildPreconditioners` have condition numbers that are not too much larger than those of its input.

LEMMA 6.3. *Let A be an* SDDM*-matrix whose largest diagonal entry is between*[3] $1/2$ *and* 1*, and let $A_1, \ldots, A_\ell$, $B_1, \ldots, B_\ell$, and $L_1, \ldots, L_\ell$ be matrices produced by* `BuildPreconditioners` *when it is run with precision u on input A. If for each i the matrix B_i is a k-ultra-sparsifier of A_{i-1}, and if*

$$\gamma_{n+1} \leq \lambda_{min}(A)/1000n^6,$$

then

(a) $k^l \leq n^4$,
(b) $\lambda_{min}(B_i) \geq \lambda_{min}(A)/2n^4$,
(c) $\lambda_{max}(B_i), \lambda_{max}(A_i) \leq 3$,
(d) $\|L_i\|_\infty \leq 3n$, *and*
(e) $\left\|L_i^{-1}\right\|_\infty \leq 2n^3/\sqrt{\lambda_{min}(A)}$.

Finally, conditions (c) *and* (d) *of Lemma* 5.3 *are satisfied by the matrices produced.*

Proof. Proposition 5.1 tells us that $\text{noff}(A_i) \leq (3h/k)\text{noff}(A_{i-1})$. As $\dim(A_i) \leq \text{noff}(A_i) + 1$ and `buildPreconditioners` stops when the dimension of A_i goes below $66h + 6$, we have

$$\ell \leq \log_{k/3h} 2m/(66h + 5) \leq \log_{k/3h} m.$$

Since $k = (14h + 1)^2$, $k/3h \geq k^{1/2}$, which implies

$$k^\ell \leq k^{\log_{k^{1/2}} m} = m^2 \leq n^4.$$

From the assumption that B_i is a k-ultra-sparsifier of A_{i-1}, we know that $B_i \preccurlyeq A_{i-1} \preccurlyeq kB_i$, and so

$$\lambda_{max}(B_i) \leq \lambda_{max}(A_{i-1})$$

and

$$\lambda_{min}(B_i) \geq (1/k)\lambda_{min}(A_{i-1}).$$

Let

$$\widehat{B}_i = PL_i \begin{pmatrix} I & 0 \\ 0 & A_i, \end{pmatrix} L_i^T P^T,$$

and let ϵ be a number satisfying

$$\epsilon \geq n\gamma_{n+1}\kappa(B_i). \tag{6.2}$$

Lemma 6.2 then implies

$$\lambda_{max}(\widehat{B}_i) \leq (1 - \epsilon)^{-1}\lambda_{max}(B_i) \leq (1 - \epsilon)^{-1}\lambda_{max}(A_{i-1}) \quad \text{and}$$
$$\lambda_{min}(\widehat{B}_i) \geq (1 - \epsilon)\lambda_{min}(B_i) \geq (1 - \epsilon)(1/k)\lambda_{min}(A_{i-1}).$$

As A_i is a Schur complement of $\widehat{B}_i$, we have the bounds [61, Corollary 2.3]

$$\lambda_{min}(\widehat{B}_i) \leq \lambda_{min}(A_i) \leq \lambda_{max}(A_i) \leq \lambda_{max}(\widehat{B}_i).$$

[3] We make assumptions about the scale of A because we are bounding the condition number of a partial Cholesky factor. We could avoid the need to make these assumptions if we instead computed partial LDL^T factorizations. The resulting algorithm would in fact be equivalent.

Recall that $\lambda_{max}(A) \leq 2 \max_i A(i,i) \leq 2$. If we set $\epsilon = 1/100\ell$ and note that $(1-\epsilon)^\ell \geq e^{-.011}$, we may now inductively prove that

$$\lambda_{max}(B_i) \leq 2(1-\epsilon)^{-(i-1)}, \quad \lambda_{min}(B_i) \geq (1-\epsilon)^{i-1}\lambda_{min}(A)/k^i,$$
$$\kappa(B_i) \leq 2(1-\epsilon)^{-2(i-1)}k^i/\lambda_{min}(A),$$

and that (6.2) is satisfied. This establishes part (b). Part (c) follows from $\lambda_{max}(A_i) \leq \lambda_{max}(\widehat{B}_i) \leq (1-\epsilon)^{-1}\lambda_{max}(B_i)$.

Each matrix L_i can be written in the form RD, where R is a lower-triangular matrix with 1s on the diagonals and D is a diagonal matrix of the form

$$\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix}.$$

The diagonal entries in D_1 are the square roots of the diagonal entries in $\widehat{B}_i$ corresponding to nodes that are eliminated by `PartialChol`. As L_i is a Cholesky factor of a diagonally dominant matrix, it is itself column diagonally dominant, and thus R is as well. As every entry of R is at most 1, $\|R\|_\infty \leq n$. By a result of Malyshev (Lemma 2 of [45]) the same is true of R^{-1}. As the diagonal entries of a symmetric matrix lie between its smallest and largest eigenvalues, the bounds we have proved on the eigenvalues of $\widehat{B}_i$ imply that its diagonals lie between $2e^{3/100}$ and $e^{-3/100}\lambda_{min}(A)/n^4$. So,

$$\|L\|_\infty \leq \|R\|_\infty \|D\|_\infty \leq 2e^{3/100}n \leq 3n$$

and

$$\left\|L^{-1}\right\|_\infty \leq \left\|R^{-1}\right\|_\infty \left\|D^{-1}\right\|_\infty \leq n \cdot n^2 e^{-3/200}/\sqrt{\lambda_{min}(A)} \leq 2n^3/\sqrt{\lambda_{min}(A)}.$$

Finally, the proof that conditions (c) and (d) of Lemma 5.3 are satisfied depends upon inequality (5.3) being satisfied. This inequality has a little bit of slack, and so it is satisfied even if

$$\lambda_{min} \geq (1-2e^{-2})/1.003 \quad \text{and} \quad \lambda_{max} \leq 1.003(1+2e^{-2})k.$$

Lemma 6.2 tells us that these conditions will hold if

$$n\gamma_{n+1}\kappa(B_i) \leq 0.003,$$

and our assumptions on γ_{n+1} guarantee that it does. □

The next lemma provides a very crude forward-error analysis of `precondCheby`. We expect that it should be possible to obtain a tighter result for a stabler variant of the preconditioned Chebyshev method.

LEMMA 6.4. *Let A be an n-by-n* SDDM *matrix of norm at most* 3. *Let B be a matrix such that $\lambda_{min}B \preccurlyeq A \preccurlyeq \lambda_{max}B$ with $\lambda_{min} \geq 1/2$ and $\lambda_{max} \leq 2k$ for $k \geq 1$. Let $\beta \geq 1$ be a number such that $\left\|B^{-1}\right\|_\infty \leq \beta$. Assume there is a number θ and a procedure f such that for all vectors y,*

$$\left\|f(y) - B^{-1}y\right\|_\infty \leq \theta \|y\|_\infty.$$

Also assume that θ and u satisfy (*recall* (6.1))

$$1/10n \geq \theta \geq 12kn\beta\gamma_n.$$

Let $\boldsymbol{x}^t$ be the result of running `precondCheby`$(A, \boldsymbol{b}, B^{-1}, t, \lambda_{min}, \lambda_{max})$ *in infinite precision, and let $\hat{\boldsymbol{x}}^t$ be the result of running* `precondCheby`$(A, \boldsymbol{b}, f, t, \lambda_{min}, \lambda_{max})$ *with precision u. Then*

$$\left\|\boldsymbol{x}^t - \hat{\boldsymbol{x}}^t\right\|_\infty \leq 17\sqrt{n}(15\beta + 5k + 1)^t \beta\theta \left\|\boldsymbol{b}\right\|_\infty .$$

Proof. Let $\boldsymbol{x}^i$ be the vector computed in the ith iteration of `precondCheby` when it is run with infinite precision. Also let $\boldsymbol{x}^0 = 0$ and let $b_0 = \left\|\boldsymbol{b}\right\|_\infty$. We have

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i - \tau_i B^{-1}(A\boldsymbol{x}^i) + \tau_i B^{-1}\boldsymbol{b}.$$

Our conditions on λ_{min} and λ_{max} imply that $\tau_i \leq 2$ for all i. So,

$$\begin{aligned}
\left\|\boldsymbol{x}^{i+1}\right\| &\leq \left\|\boldsymbol{x}^i\right\| + 2\left\|B^{-1}A\right\|\left\|\boldsymbol{x}^i\right\| + 2\left\|B^{-1}\boldsymbol{b}\right\| \\
&\leq 5k\left\|\boldsymbol{x}^i\right\| + 2\left\|B^{-1}\boldsymbol{b}\right\| \\
&\leq (5k+1)^i 2\left\|B^{-1}\boldsymbol{b}\right\| && \text{by induction.}
\end{aligned}$$

Using standard relations between the 2- and ∞-norms, we conclude

$$\left\|\boldsymbol{x}^i\right\|_\infty \leq 2(5k+1)^{i-1}\sqrt{n}\beta b_0.$$

In the rest of the proof, we make liberal use of relations such as these between norms, as well as the following inequalities:

$\left\|A\right\|_\infty \leq 6$, which follows from the diagonal dominance of A and $\left\|A\right\| \leq 3$, and $\left\|B^{-1}\right\|_\infty \geq 1/6\sqrt{n}$, which follows from $B \preccurlyeq 2A$ and $\left\|B^{-1}\right\|_\infty \geq \left\|B^{-1}\right\|/\sqrt{n}$.

We now set

$$\begin{aligned}
\boldsymbol{y}^i &= A\boldsymbol{x}^i, \\
\boldsymbol{z}^i &= B^{-1}(\boldsymbol{y}^i),
\end{aligned}$$

and we let $\hat{\boldsymbol{y}}^i$ and $\hat{\boldsymbol{z}}^i$ be the analogous quantities computed using precision u and the function f instead of B^{-1}.

We compute

$$\begin{aligned}
\left\|\hat{\boldsymbol{y}}^i - \boldsymbol{y}^i\right\|_\infty &\leq \gamma_n \left\|A\right\|_\infty \left\|\hat{\boldsymbol{x}}^i\right\|_\infty + \left\|A\right\|_\infty \left\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\right\|_\infty \quad \text{(following [33, section 3.5])} \\
&\leq \gamma_n \left\|A\right\|_\infty \left\|\boldsymbol{x}^i\right\|_\infty + \gamma_n \left\|A\right\|_\infty \left\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\right\|_\infty + \left\|A\right\|_\infty \left\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\right\|_\infty \\
&\leq 6\gamma_n \|\boldsymbol{x}^i\|_\infty + 7\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty, \quad \text{as } \left\|A\right\|_\infty \leq 6 \text{ and } \gamma_n \leq 1/6.
\end{aligned}$$

We then compute

$$\begin{aligned}
\|\hat{\boldsymbol{z}}^i - \boldsymbol{z}^i\|_\infty &= \|f(\hat{\boldsymbol{y}}^i) - B^{-1}\boldsymbol{y}^i\|_\infty \\
&\leq \|f(\hat{\boldsymbol{y}}^i) - B^{-1}\hat{\boldsymbol{y}}^i\|_\infty + \|B^{-1}\hat{\boldsymbol{y}}^i - B^{-1}\boldsymbol{y}^i\|_\infty \quad \text{by the triangle inequality} \\
&\leq \theta\|\hat{\boldsymbol{y}}^i\|_\infty + \beta\|\hat{\boldsymbol{y}}^i - \boldsymbol{y}^i\|_\infty \\
&\leq \theta\|\boldsymbol{y}^i\|_\infty + (\theta+\beta)\|\hat{\boldsymbol{y}}^i - \boldsymbol{y}^i\|_\infty \\
&\leq \theta\|\boldsymbol{y}^i\|_\infty + 2\beta\|\hat{\boldsymbol{y}}^i - \boldsymbol{y}^i\|_\infty && \text{as } \|B^{-1}\|_\infty \geq 1/6n \geq \theta.
\end{aligned}$$

If we now substitute our upper bound on $\|\hat{\boldsymbol{y}}^i - \boldsymbol{y}^i\|_\infty$ and apply the inequality $\|\boldsymbol{y}^i\|_\infty \leq \|A\|_\infty \|\boldsymbol{x}^i\|_\infty$, we obtain

$$\begin{aligned}\|\hat{\boldsymbol{z}}^i - \boldsymbol{z}^i\|_\infty &\leq \theta \|A\|_\infty \|\boldsymbol{x}^i\|_\infty + 12\beta\gamma_n \|\boldsymbol{x}^i\|_\infty + 14\beta \|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty \\ &\leq 7\theta \|\boldsymbol{x}^i\|_\infty + 14\beta \|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty, \quad \text{as } \theta \geq 12\beta\gamma_n \text{ and } \|A\|_\infty \leq 6.\end{aligned}$$

Finally, we find

$$\begin{aligned}\|\hat{\boldsymbol{x}}^{i+1} - \boldsymbol{x}^{i+1}\|_\infty \leq (1+\gamma_3)\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty + (1+\gamma_3)\|\hat{\boldsymbol{z}}^i - \boldsymbol{z}^i\|_\infty \\ + \gamma_3\big(\|\boldsymbol{z}^i\|_\infty + \|\boldsymbol{x}^i\|_\infty + \beta b_0\big) + \theta b_0,\end{aligned}$$

where the terms involving γ_3 account for the imprecision introduced when computing the sum of the three vectors in the Chebyshev recurrence. Using the upper bound on $\|\hat{\boldsymbol{z}}^i - \boldsymbol{z}^i\|_\infty$ and the inequality $\|\boldsymbol{z}^i\|_\infty \leq \|B^{-1}A\|_\infty \|\boldsymbol{x}^i\|_\infty \leq 2k\sqrt{n}\|\boldsymbol{x}^i\|_\infty$, we see that this last expression is at most

$$(1+\gamma_3)(1+14\beta)\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty + \big((1+\gamma_3)7\theta + \gamma_3(1+2k\sqrt{n})\big)\|\boldsymbol{x}^i\|_\infty + (\gamma_3\beta + \theta)b_0.$$

We simplify this expression by observing that

$$\begin{aligned}(1+\gamma_3)(1+14\beta) &\leq 15\beta, \\ (1+\gamma_3)7\theta + \gamma_3(1+2k\sqrt{n}) &\leq 8\theta,\end{aligned}$$

and

$$\begin{aligned}8\theta\|\boldsymbol{x}^i\|_\infty + (\gamma_3\beta+\theta)b_0 &\leq 16\theta(5k+1)^{i-1}\sqrt{n}\beta b_0 + 2\beta b_0 \quad (\text{as } \gamma_3\beta + \theta \leq 2\beta) \\ &\leq 17\theta\sqrt{n}(5k+1)^{i-1}\beta b_0.\end{aligned}$$

We conclude that

$$\|\hat{\boldsymbol{x}}^{i+1} - \boldsymbol{x}^{i+1}\|_\infty \leq 15\beta\|\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i\|_\infty + 17\theta\sqrt{n}(5k+1)^{i-1}\beta b_0.$$

As $\hat{\boldsymbol{x}}^0 = \boldsymbol{x}^0 = \boldsymbol{0}$, we may apply this inequality inductively to obtain

$$\left\|\hat{\boldsymbol{x}}^t - \boldsymbol{x}^t\right\| \leq 17(15\beta + 5k + 1)^t \theta\sqrt{n}\beta b_0. \qquad \square$$

We now establish analogous bounds on the stability of $\texttt{solve}_{B_i}$.

LEMMA 6.5. *Assume that the largest diagonal entry of A is between $1/2$ and 1. Let $A_1, \ldots, A_\ell$, $B_1, \ldots, B_\ell$, and $L_1, \ldots, L_\ell$ be as in Lemma 6.3. Let i be less than ℓ and let f_{i+1} be a function such that for all vectors $\boldsymbol{b}$,*

$$\left\|f_{i+1}(\boldsymbol{b}) - \texttt{solve}_{B_{i+1}}(\boldsymbol{b})\right\|_\infty \leq \theta \|\boldsymbol{b}\|_\infty.$$

Let $\boldsymbol{x}$ be the result of running $\texttt{solve}_{B_i}$ on input $\boldsymbol{b}$ with full precision, and let $\hat{\boldsymbol{x}}$ be the result of running $\texttt{solve}_{B_i}$ on input $\boldsymbol{b}$ with precision u, using f_{i+1} in place of $\texttt{solve}_{B_{i+1}}$. There exist constants d_1, d_2, d_3, and d_4 so that if

$$\gamma_n \leq \theta\lambda_{min}^2(A)/50n^8 \quad \textit{and} \quad \theta \leq 1/d_3(n\kappa(A))^{d_4},$$

then

$$\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_\infty \leq \theta d_1 (n\kappa(A))^{d_2\sqrt{k}} \|\boldsymbol{x}\|_\infty.$$

The same bound holds for $i = \ell$ if $\texttt{solve}_{B_\ell}$ is executed using precision u, assuming that Z_ℓ is applied by forward and backward substitution through Cholesky factors of A_ℓ.

Proof. We first consider the case when $i < \ell$. Let $\boldsymbol{s}, \boldsymbol{s}_0, \boldsymbol{s}_1$, and $\boldsymbol{y}$ denote the vectors computed by $\texttt{solve}_{B_i}$ when it runs in full precision. Similarly, let $\hat{\boldsymbol{s}}, \hat{\boldsymbol{s}}_0, \hat{\boldsymbol{s}}_1$, and $\hat{\boldsymbol{y}}$ be the corresponding vectors computed when $\texttt{solve}_{B_i}$ runs with precision u and uses f. By inequality (8.2) of Higham [33], we have

$$\|\hat{\boldsymbol{s}} - \boldsymbol{s}\|_\infty \leq \frac{\mathrm{Cond}(L_i)\gamma_n}{1 - \mathrm{Cond}(L_i)\gamma_n} \|\boldsymbol{s}\|_\infty , \tag{6.3}$$

where

$$\mathrm{Cond}(L_i) \leq \|L_i\|_\infty \left\|L_i^{-1}\right\|_\infty .$$

By Lemma 6.3, we know that this product is at most $6n^4/\sqrt{\lambda_{min}(A)}$. We have assumed γ_n is small enough so that this gives

$$\|\hat{\boldsymbol{s}} - \boldsymbol{s}\|_\infty \leq 7n^4\gamma_n \|\boldsymbol{s}\|_\infty / \sqrt{\lambda_{min}(A)}.$$

We also know that

$$\|\boldsymbol{s}\|_\infty \leq \left\|L_i^{-1}\right\|_\infty \|\boldsymbol{b}\|_\infty \leq 2n^3 \|\boldsymbol{b}\|_\infty / \sqrt{\lambda_{min}(A)},$$

and so

$$\|\hat{\boldsymbol{s}}_1 - \boldsymbol{s}_1\|_\infty \leq \|\hat{\boldsymbol{s}} - \boldsymbol{s}\|_\infty \leq 12n^7\gamma_n \|\boldsymbol{b}\|_\infty / \lambda_{min}(A).$$

Again applying our assumptions on γ_n, we derive

$$\|\hat{\boldsymbol{s}}_1\|_\infty \leq \|\hat{\boldsymbol{s}}\|_\infty \leq 3n^3 \|\boldsymbol{b}\|_\infty / \sqrt{\lambda_{min}(A)}.$$

We now examine the relationship between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. For $i < \ell$, Lemma 6.4 tells us that

$$\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_\infty \leq \alpha\theta \|\hat{\boldsymbol{s}}_1\|_\infty + \left\|\texttt{solve}_{B_{i+1}}\right\|_\infty \|\hat{\boldsymbol{s}}_1 - \boldsymbol{s}_1\|_\infty ,$$

where

$$\alpha = 17(15\beta + 5k + 1)^t \theta\sqrt{n}\beta$$

and

$$\beta = \left\|\texttt{solve}_{B_{i+1}}\right\|_\infty \leq \sqrt{n} \left\|\texttt{solve}_{B_{i+1}}\right\| .$$

By part (d) of Lemma 5.3 and by Lemma 6.3, we know

$$\left\|\texttt{solve}_{B_{i+1}}\right\| \leq 2(1 + 2e^{-2}) \left\|B_{i+1}^{-1}\right\| \leq 6n^4/\lambda_{min}(A).$$

From the specification of $\texttt{solve}_{B_i}$, we have $t = \lceil 1.33\sqrt{k} \rceil$. So,

$$\alpha \leq d_1(n\kappa(A))^{d_2\sqrt{k}}$$

for some constants d_1 and d_2. From our assumption on the relation between θ and γ_n we conclude

$$\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_\infty \leq 2\alpha\theta \|\hat{\boldsymbol{s}}\|_\infty .$$

We also know that

$$\|\boldsymbol{y}\|_\infty \leq \left\|\mathtt{solve}_{B_{i+1}}\right\|_\infty \|\boldsymbol{s}\|_\infty \leq 12n^{7.5} \|\boldsymbol{b}\|_\infty / \lambda_{min}^{3/2}(A)$$

and by our assumptions on θ that

$$\|\hat{\boldsymbol{y}}\|_\infty \leq 13n^{7.5} \|\boldsymbol{b}\|_\infty / \lambda_{min}^{3/2}(A).$$

There are two sources of discrepancy between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$: error from multiplying $\hat{\boldsymbol{y}}$ by L_i^{-T} and error from the difference between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$. Let $\tilde{\boldsymbol{x}} = L_i^{-T}\hat{\boldsymbol{y}}$. We have

$$\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_\infty \leq \|\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|_\infty + \|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_\infty .$$

As with the derivation following (6.3), we find

$$\begin{aligned}\|\hat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|_\infty &\leq 7n^4\gamma_n \|\tilde{\boldsymbol{x}}\|_\infty / \sqrt{\lambda_{min}(A)} \\ &\leq 7n^4\gamma_n \left\|L_i^{-T}\right\|_\infty \|\hat{\boldsymbol{y}}\|_\infty / \sqrt{\lambda_{min}(A)}.\end{aligned}$$

We also have

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_\infty \leq \left\|L_i^{-T}\right\|_\infty \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_\infty .$$

We conclude that

$$\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_\infty \leq \theta d_1 (n\kappa(A))^{d_2\sqrt{k}} \|\boldsymbol{b}\|_\infty$$

for some other constants d_1 and d_2. Finally, we prove the main claim of the lemma for some different constants d_1 and d_2 by observing that

$$\|\boldsymbol{b}\|_\infty \leq \left\|\mathtt{solve}_{B_i}^{-1}\right\|_\infty \|\boldsymbol{x}\|_\infty$$

and that

$$\left\|\mathtt{solve}_{B_i}^{-1}\right\|_\infty \leq \sqrt{n}\left\|\mathtt{solve}_{B_i}^{-1}\right\| \leq (1-2e^{-2})^{-1}\sqrt{n}\|B_i\| \leq 5\sqrt{n},$$

where the second inequality follows from part (d) of Lemma 5.3 and the last inequality follows from part (c) of Lemma 6.3.

In the case $i = \ell$, we may apply the same analysis but without the need to account for a call to `precondCheby`. While there is no function f_{i+1} (or we can view it as the identity), we still require the upper bound induced on γ_n through θ. □

THEOREM 6.6. *Let A be an* SDDM*-matrix whose largest diagonal entry is between $1/2$ and 1. Let $\epsilon < 1/2$, let $\boldsymbol{x}$ be the result of running* $\mathtt{solve}(A, \boldsymbol{b}, \epsilon)$ *in infinite precision, and let $\hat{\boldsymbol{x}}$ be the result of running this routine and all its subroutines with precision u. There exist constants d_1, d_2, and d_3 so that if*

$$u \leq (\epsilon / d_1\kappa(A))^{d_2 \log^{d_3} n},$$

then

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_\infty \leq \epsilon \|\boldsymbol{x}\|_\infty .$$

That is, it suffices to run `solve` with $O(\log(\kappa(A)/\epsilon)\log^{d_3} n)$ bits of precision.

Proof. As `solve` uses `precondCheby` to solve systems in A by using B_1 as a preconditioner, we begin by examining the requirements on $\texttt{solve}_{B_1}$.

For each i, let $f_i(\boldsymbol{b}_i)$ be the output of $\texttt{solve}_{B_i}$ when it and all its subroutines are executed with precision u on input $\boldsymbol{b}_i$. We need to establish that there exist numbers θ_i that satisfy the conditions of Lemma 6.5 for f_i (where in the statement of that lemma $\texttt{solve}_{B_i}$ is treated as the infinite-precision linear operator). We may assume that the constants d_1, d_2, d_3, and d_4 from Lemma 6.5 are all at least 1. So, it suffices to choose θ_1 less than $1/d_3(n\kappa(A))^{d_4}$,

$$\theta_i = \frac{\theta_1}{d_1(n\kappa(A))^{d_2\sqrt{k}(i-1)}},$$

and u so that

$$\gamma_n \leq \theta_\ell \lambda_{max}^2(A)/50n^8.$$

If we now apply the analysis of `precondCheby` from Lemma 6.4, assuming that the constants d_3 and d_4 are big enough that $1/d_3(n\kappa(A))^{d_4} < 1/10n$, we see that

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_\infty \leq 17\sqrt{n}(15\beta + 5k + 1)^t \beta\theta_1 \|\boldsymbol{b}\|_\infty,$$

where

$$\beta = \|\texttt{solve}_{B_1}\|_\infty$$

and

$$t \leq \sqrt{k}\ln(2/\epsilon).$$

As in the proof of Lemma 6.5, we can show that

$$\|\texttt{solve}_{B_1}\|_\infty \leq 6n^4/\lambda_{min}(A)$$

and

$$\|\boldsymbol{b}\|_\infty \leq 5\sqrt{n}\|\boldsymbol{x}\|_\infty.$$

To finish the proof, we observe that $\ell \leq \log_2 n$ and recall from (5.1) that $\sqrt{k} = (14h+1)$ and that h was set in (1.1) to $c_3 \log_2^{c_4} n$, where c_3 and c_4 were parameters related to the quality of the ultra-sparsifiers that we could produce. We thus conclude that we can find (different) constants d_1, d_2, and d_3 that satisfy the claims of the theorem. ∎

7. Computing approximate Fiedler vectors. As an example of the use of SDD_0 systems, we now show that our solver immediately leads to a nearly linear time approximation of the eigenvector associated with the second-smallest eigenvalue of the Laplacian matrix of a weighted graph. Fiedler [24] was the first to recognize this vector could be used to partition the underlying graph. From a result of Mihail [46], we know that any vector whose Rayleigh quotient is close to this eigenvalue can also be used to find a good partition. We call such a vector an *approximate Fiedler vector.*

DEFINITION 7.1 (approximate Fiedler vector). *For a Laplacian matrix A, $\boldsymbol{v}$ is an ϵ-approximate Fiedler vector if $\boldsymbol{v}$ is orthogonal to the all-1's vector and*

$$\frac{\boldsymbol{v}^T A \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{v}} \leq (1+\epsilon)\lambda_2(A),$$

where $\lambda_2(A)$ is the second-smallest eigenvalue of A (which is also referred to as the Fiedler value of the graph defined by A).

Our linear system solvers may be used to quickly compute ϵ-approximate Fiedler vectors. The algorithm `ApproxFiedler` does so with probability at least $1-p$. This algorithm works by applying the inverse power method to a random initial vector and by using `solve` to accomplish the inversion. As there is some chance that this algorithm could fail, or that `BuildPreconditioners` could fail, we run the entire process $\log_2 1/p$ times and return the best result.

$\boldsymbol{v} = \texttt{ApproxFiedler}(A, \epsilon, p)$

1. Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1 + 2e^{-2})k$ and $t = \lceil 0.67\sqrt{k}\ln(8/\epsilon)\rceil$.
2. Set $s = 8\ln(18(n-1)/\epsilon)/\epsilon$.
3. For $a = 1, \ldots, \lceil \log_2 1/p \rceil$.
   (a) Run $\texttt{BuildPreconditioners}(A)$.
   (b) Choose $\boldsymbol{r}^0$ to be a uniform random unit vector orthogonal to the all-1's vector.
   (c) For $b = 1, \ldots, s$
   $$\boldsymbol{r}^b = \texttt{precondCheby}(A, \boldsymbol{r}^{b-1}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max}).$$
   (d) Set $\boldsymbol{v}_a = \boldsymbol{r}^s$.
4. Let a_0 be the index of the vector minimizing $\boldsymbol{v}_{a_0}^T A \boldsymbol{v}_{a_0} / \boldsymbol{v}_{a_0}^T \boldsymbol{v}_{a_0}$.
5. Set $\boldsymbol{v} = \boldsymbol{v}_{a_0}$.

THEOREM 7.2. *On input a Laplacian matrix A with m nonzero entries and $\epsilon, p > 0$, with probability at least $1-p$, $\texttt{ApproxFiedler}(A, \epsilon, p)$ computes an ϵ-approximate Fiedler vector of A in time*

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

for some constant c.

Our proof of Theorem 7.2 will use the following proposition.

PROPOSITION 7.3. *If Z is a matrix such that*

$$(1-\epsilon)Z^\dagger \preccurlyeq A \preccurlyeq (1+\epsilon)Z^\dagger,$$

and $\boldsymbol{v}$ is a vector orthogonal to the all-1's vector such that $\boldsymbol{v}^T Z^\dagger \boldsymbol{v} \leq (1+\epsilon)\lambda_2(Z^\dagger)$, for some $\epsilon \leq 1/5$, then $\boldsymbol{v}$ is a 4ϵ-approximate Fiedler vector of A.

Proof. We first observe that

$$\lambda_2(Z^\dagger) \leq \lambda_2(A)/(1-\epsilon).$$

We then compute

$$\begin{aligned}
\boldsymbol{v}^T A \boldsymbol{v} &\leq (1+\epsilon)\boldsymbol{v}^T Z^\dagger \boldsymbol{v} \\
&\leq (1+\epsilon)(1+\epsilon)\lambda_2(Z^\dagger) \\
&\leq (1+\epsilon)(1+\epsilon)\lambda_2(A)/(1-\epsilon) \\
&\leq (1+4\epsilon)\lambda_2(A)
\end{aligned}$$

for $\epsilon \leq 1/5$. ∎

Proof of Theorem 7.2. As we did in the proof of Lemma 5.3 and Theorem 5.5, we can show that $\texttt{precondCheby}(A, \boldsymbol{b}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max})$ is a linear operator in $\boldsymbol{b}$. Let Z denote the matrix realizing this operator. As in the proof of Lemma 5.3, we can show that $(1-\epsilon/4)Z^\dagger \preccurlyeq A \preccurlyeq (1+\epsilon/4)Z^\dagger$.

By Proposition 7.3, it suffices to show that with probability at least 1/2 each vector $\boldsymbol{v}_a$ satisfies

$$\boldsymbol{v}_a^T Z^\dagger \boldsymbol{v}_a / \boldsymbol{v}_a^T \boldsymbol{v}_a \leq (1+\epsilon/4)\lambda_2(Z^\dagger).$$

To this end, let $0 = \mu_1 \leq \mu_2 \leq \cdots \leq \mu_n$ be the eigenvalues of $Z^\dagger$, and let $\mathbf{1} = \boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be corresponding eigenvectors. Let

$$\boldsymbol{r}^0 = \sum_{i \geq 2} \alpha_i \boldsymbol{u}_i,$$

and recall that (see, e.g., [48, Lemma B.1])

$$\Pr\left[|\alpha_2| \geq 2/3\sqrt{(n-1)}\right] \geq \frac{2}{\sqrt{2\pi}} \int_{2/3}^{\infty} e^{-t^2/2}\, dt \geq 0.504.$$

Thus, with probability at least 1/2, the call to `BuildPreconditioners` succeeds and $|\alpha_2| \geq 2/3\sqrt{(n-1)}$. In this case,

$$s \geq 8 \ln(8/\alpha_2^2 \epsilon)/\epsilon. \tag{7.1}$$

We now show that this inequality implies that $\boldsymbol{r}^s$ satisfies

$$\frac{(\boldsymbol{r}^s)^T Z^\dagger \boldsymbol{r}^s}{(\boldsymbol{r}^s)^T \boldsymbol{r}^s} \leq (1+\epsilon/4)\mu_2.$$

To see this, let j be the greatest index such that $\mu_j \leq (1+\epsilon/8)\mu_2$, and compute

$$\boldsymbol{r}^s = Z^s \boldsymbol{r}^0 = \sum_{i \geq 2} \boldsymbol{u}_i \alpha_i / \mu_i^s,$$

so

$$\begin{aligned}
\frac{(\boldsymbol{r}^s)^T Z^\dagger \boldsymbol{r}^s}{(\boldsymbol{r}^s)^T \boldsymbol{r}^s} &= \frac{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2s-1}}{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2s}} \\
&\leq \frac{\sum_{j \geq i \geq 2} \alpha_i^2/\mu_i^{2s-1}}{\sum_{j \geq i \geq 2} \alpha_i^2/\mu_i^{2s}} + \frac{\sum_{i>j} \alpha_i^2/\mu_i^{2s-1}}{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2s}} \\
&\leq \mu_j + \frac{\sum_{i>j} \alpha_i^2/\mu_i^{2s-1}}{\alpha_2^2/\mu_2^{2s}} \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \left(\frac{\sum_{i>j} \alpha_i^2 (\mu_2/\mu_i)^{2s-1}}{\alpha_2^2} \right) \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \left(\frac{\sum_{i>j} \alpha_i^2 (1/(1+\epsilon/8))^{2s-1}}{\alpha_2^2} \right) \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \sum_{i>j} \alpha_i^2 \epsilon/8 \qquad \text{(by inequality (7.1))} \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2(\epsilon/8) \\
&\leq (1+\epsilon/4)\mu_2. \qquad \square
\end{aligned}$$

Note that our algorithm is asymptotically effective when ϵ is not too small. For example, the nearly linear time approximation of the Fiedler vector has an immediate application in graph partitioning. Particularly, our algorithm (by setting $\epsilon = 1/2$) together with Mihail's extension [46] of the Cheeger's inequality yields the following algorithmic result.

COROLLARY 7.4 (nearly linear time ratio cut). *For any undirected graph G of n vertices and m edges, with probability at least $1-p$ and in time $O(m \log^c m \log(1/p))$, one can compute a set $S \subset V$ such that $|S| \leq |V|/2$, and*

$$\frac{|E(S, V-S)|}{|S|} \leq \sqrt{3\Delta_{\max}\lambda_2},$$

where λ_2 is the Fiedler value of G, $E(S, V-S)$ denotes the set of edges between S and $V-S$, $\Delta_{\max}$ denotes the maximum degree of G, and c is the constant as defined in Theorem 7.2.

8. Laplacians and weighted graphs. We will find it convenient to describe and analyze our preconditioners for Laplacian matrices in terms of weighted graphs. This is possible because of the isomorphism between Laplacian matrices and weighted graphs. To an n-by-n Laplacian matrix A, we associate the graph with vertex set $\{1, \ldots, n\}$ having an edge between vertices u and v of weight $-A(u,v)$ for each u and v such that $A(u,v)$ is nonzero.

All the graphs we consider in this paper will be weighted. If u and v are distinct vertices in a graph, we write $\mathbf{(}u, v\mathbf{)}$ to denote an edge between u and v of weight 1. Similarly, if $w > 0$, then we write $w\mathbf{(}u, v\mathbf{)}$ to denote an edge between u and v of weight w. A weighted graph is then a pair $G = (V, E)$, where V is a set of vertices and E is a set of weighted edges on V, each of which spans a distinct pair of vertices. The Laplacian matrix L_G of the graph G is the matrix such that

$$L_G(u,v) = \begin{cases} -w & \text{if there is an edge } w\mathbf{(}u,v\mathbf{)} \in E, \\ 0 & \text{if } u \neq v \text{ and there is no edge between } u \text{ and } v \text{ in } E, \\ \sum_{w\mathbf{(}u,x\mathbf{)} \in E} w & \text{if } u = v. \end{cases}$$

We recall that for every vector $\boldsymbol{x} \in \mathbb{R}^n$,

$$\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{w\mathbf{(}u,v\mathbf{)} \in E} w(\boldsymbol{x}_u - \boldsymbol{x}_v)^2.$$

For graphs G and H on the same set of vertices, we define the graph $G + H$ to be the graph whose Laplacian matrix is $L_G + L_H$.

9. Graphic inequalities, resistance, and low-stretch spanning trees. In this section, we introduce the machinery of "graphic inequalities" that underlies the proofs in the rest of the paper. We then introduce low-stretch spanning trees and use graphic inequalities to bound how well a low-stretch spanning tree preconditions a graph. This proof provides the motivation for the construction in the next section.

We begin by overloading the notation $\preccurlyeq$ by writing

$$G \preccurlyeq H \qquad \text{or} \qquad E \preccurlyeq F$$

if $G = (V, E)$ and $H = (V, F)$ are two graphs such that their Laplacian matrices, L_G and L_H satisfy

$$L_G \preccurlyeq L_H.$$

Many facts that have been used in the chain of work related to this paper can be simply expressed with this notation. For example, the splitting lemma of [9] becomes

$$A_1 \preccurlyeq B_1 \quad \text{and} \quad A_2 \preccurlyeq B_2 \quad \text{implies} \quad A_1 + A_2 \preccurlyeq B_1 + B_2.$$

We also observe that if B is a subgraph of A, then

$$B \preccurlyeq A.$$

We define the *resistance* of an edge to be the reciprocal of its weight. Similarly, we define the resistance of a simple path to be the sum of the resistances of its edges. For example, the resistance of the path $w_1\mathbf{(}1,2\mathbf{)}, w_2\mathbf{(}2,3\mathbf{)}, w_3\mathbf{(}3,4\mathbf{)}$ is $(1/w_1+1/w_2+1/w_3)$. Of course, the resistance of a trivial path with one vertex and no edges is zero. If one multiplies all the weights of the edges in a path by α, its resistance decreases by a factor of α.

The next lemma says that a path of resistance r supports an edge of resistance r. This lemma may be derived from the rank-one support lemma of [13], and appears in simpler form as the congestion-dilation lemma of [9] and Lemma 4.6 of [30]. We present a particularly simple proof.

LEMMA 9.1 (path inequality). *Let $e = w\mathbf{(}u,v\mathbf{)}$ and let P be a path from u to v. Then,*

$$e \preccurlyeq w \ \textit{resistance}(P) \cdot P.$$

Proof. After dividing both sides by w, it suffices to consider the case $w = 1$. Without loss of generality, we may assume that $e = \mathbf{(}1, k+1\mathbf{)}$ and that P consists of the edges $w_i\mathbf{(}i, i+1\mathbf{)}$ for $1 \le i \le k$. In this notation, the lemma is equivalent to

$$\mathbf{(}1,k+1\mathbf{)} \preccurlyeq \left(\sum_i \frac{1}{w_i}\right)\left(w_1\mathbf{(}1,2\mathbf{)} + w_2\mathbf{(}2,3\mathbf{)} + \cdots + w_k\mathbf{(}k,k+1\mathbf{)}\right).$$

We prove this for the case $k = 2$. The general case follows by induction.

Recall Cauchy's inequality, which says that for all $0 < \alpha < 1$,

$$(a+b)^2 \le a^2/\alpha + b^2/(1-\alpha).$$

For $k = 2$, the lemma is equivalent to the statement that for all $\boldsymbol{x} \in \mathbb{R}^3$,

$$\boldsymbol{x}^T L_{\mathbf{(}1,3\mathbf{)}} \boldsymbol{x} \le \left(\frac{1}{w_1} + \frac{1}{w_2}\right) \boldsymbol{x}^T \left(L_{w_1\mathbf{(}1,2\mathbf{)}} + L_{w_2\mathbf{(}2,3\mathbf{)}}\right) \boldsymbol{x}.$$

This is equivalent to

$$(x_1 - x_3)^2 \le (1 + w_1/w_2)(x_1 - x_2)^2 + (1 + w_2/w_1)(x_2 - x_3)^2,$$

which follows from Cauchy's inequality with $\alpha = w_2/(w_1 + w_2)$. ∎

Recall that a *spanning tree* of a weighted graph $G = (V, E)$ is a connected subgraph of G with exactly $|V|-1$ edges. The weights of edges that appear in a spanning tree are assumed to be the same as in G. If T is a spanning tree of a graph $G = (V, E)$, then for every pair of vertices $u, v \in V$, T contains a unique path from u to v. We let $T(u, v)$ denote this path. We now use graphic inequalities to derive a bound on how

well T preconditions G. This bound strengthens a bound of Boman and Hendrickson [13, Lemma 4.9].

LEMMA 9.2 (tree preconditioners). *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. Then,*

$$T \preccurlyeq G \preccurlyeq \left(\sum_{e \in E} \frac{\mathit{resistance}(T(e))}{\mathit{resistance}(e)} \right) \cdot T.$$

Proof. As T is a subgraph of G, $T \preccurlyeq G$ is immediate. To prove the right-hand inequality, we compute

$$\begin{aligned} E &= \sum_{e \in E} e \\ &\preccurlyeq \sum_{e \in E} \frac{\text{resistance}(T(e))}{\text{resistance}(e)} \cdot T(e) && \text{by Lemma 9.1} \\ &\preccurlyeq \left(\sum_{e \in E} \frac{\text{resistance}(T(e))}{\text{resistance}(e)} \right) \cdot T && \text{as } T(e) \preccurlyeq T. \end{aligned}$$

□

DEFINITION 9.3 (stretch). *Given a tree T spanning a set of vertices V and a weighted edge $e = w\big(u, v\big)$ with $u, v \in V$, we define the stretch of e with respect to T to be*

$$\mathit{st}_T(e) = \frac{\mathit{resistance}(T(e))}{\mathit{resistance}(e)} = w \cdot \mathit{resistance}(T(e)).$$

If E is a set of edges on V, then we define

$$\mathit{st}_T(E) = \sum_{e \in E} \mathit{st}_T(e).$$

With this definition, the statement of Lemma 9.2 may be simplified to

$$T \preccurlyeq G \preccurlyeq \text{st}_T(E) \cdot T. \tag{9.1}$$

We will often use the following related inequality, which follows immediately from Lemma 9.1 and the definition of stretch.

$$w\big(u, v\big) \preccurlyeq \text{st}_T(w\big(u, v\big))\ T(u, v) = w\ \text{st}_T(\big(u, v\big))\ T(u, v). \tag{9.2}$$

10. Preconditioning with augmented low-stretch trees. In the remainder of the paper, we will build ultra-sparsifers for weighted graphs to provide preconditioners for their Laplacian matrices. Our construction starts with a low-stretch spanning tree. By Theorem 10.1, every weighted graph G has a spanning tree T with stretch $O(m \log n \log\log n)$. To focus our discussion on nonlogarithmic factors, we let $\tilde{O}(f(n))$ denote $O(f(n) \log^c n)$ for some constant c. Then, by (9.1), we have

$$T \preccurlyeq G \preccurlyeq \tilde{O}(m) \cdot T.$$

Our goal, for integer $0 < t \leq m$, is to construct a set of edges $F \subset E$ such that $|F| = \tilde{O}(t)$, and

$$(T \cup F) \preccurlyeq G \preccurlyeq \tilde{O}\left(\frac{m}{t}\right) \cdot (T \cup F). \tag{10.1}$$

We will achieve (10.1) in two steps. In this section, we present a simple algorithm, `UltraSimple`, that works by simply adding edges to low-stretch spanning trees. This algorithm is sufficient to obtain all our results for planar graphs. For arbitrary graphs, this algorithm might add too many additional edges: As shown in Theorem 10.5, the algorithm uses $O(t^2)$ edges to achieve (10.1). We will then show in section 11 how these extra edges can be removed via sparsification.

10.1. Low-stretch trees. Low-stretch spanning trees were introduced by Alon et al. [3]. At present, the construction of spanning trees with the lowest stretch is due to Abraham and Neiman [2], who prove the following.

THEOREM 10.1 (low-stretch spanning trees). *There exists an $O(m \log n \log\log n)$-time algorithm,* `LowStretch`, *that on input of a weighted connected graph $G = (V, E)$ outputs a spanning tree T of G such that*

$$st_T(E) \leq c_{AN}\ m \log n \log\log n,$$

where $m = |E|$, for some constant c_{AN}.

10.2. Augmenting low-stretch spanning trees. Our procedure for deciding which edges to add to a tree begins by decomposing the tree into subtrees. In the decomposition, we allow subtrees to overlap at a single vertex or even to consist of just a single vertex. Then, for every pair of subtrees connected by edges of E, we add one such edge of E to the tree. The subtrees are specified by the subset of the vertices that they span.

DEFINITION 10.2. *Given a tree T that spans a set of vertices V, a T-decomposition is a decomposition of V into sets $W_1, \ldots, W_h$ such that $V = \cup W_i$, the graph induced by T on each W_i is a tree, possibly with just one vertex, and for all $i \neq j$, $|W_i \cap W_j| \leq 1$.*

Given an additional set of edges E on V, a (T, E)-decomposition is a pair $(\{W_1, \ldots, W_h\}, \rho)$, where $\{W_1, \ldots, W_h\}$ is a T-decomposition and ρ is a map that sends each edge of E to a set or pair of sets in $\{W_1, \ldots, W_h\}$ so that for each edge in $(u, v) \in E$,

(a) *if $\rho(u, v) = \{W_i\}$, then $\{u, v\} \subseteq W_i$, and*
(b) *if $\rho(u, v) = \{W_i, W_j\}$, then either $u \in W_i$ and $v \in W_j$, or $u \in W_j$, and $v \in W_i$.*

We remark that as the sets W_i and W_j can overlap, it is possible that $\rho(u, v) = \{W_i, W_j\}$, $u \in W_i$, and $v \in W_i \cap W_j$. See Figure 2 for an example.

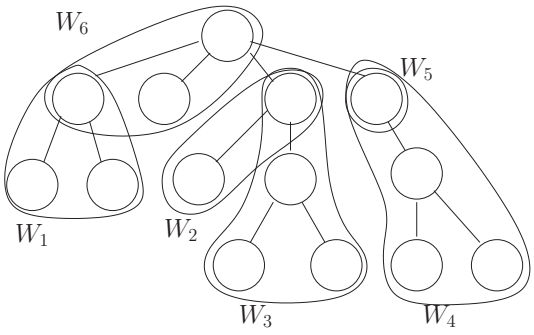

FIG. 2. *An example of a tree decomposition. Note that sets W_1 and W_6 overlap and that set W_5 is a singleton set and that it overlaps W_4.*

We use the following tree decomposition theorem to show that one can always quickly find a T-decomposition of E with few components in which the sum of stretches of the edges attached to each nonsingleton component is not too big. As the theorem holds for any nonnegative function η on the edges, not just stretch, we state it in this general form.

THEOREM 10.3 (`decompose`). *There exists a linear time algorithm, which we invoke with the syntax*

$$(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t),$$

that on input a set of edges E on a vertex set V, a spanning tree T on V, a function $\eta : E \to \mathbb{R}^+$, and an integer $1 < t \leq \sum_{e \in E} \eta(e)$ outputs a (T, E)-decomposition $(\{W_1, \ldots, W_h\}, \rho)$ such that

(a) $h \leq t$,

(b) *for all W_i such that $|W_i| > 1$,*

$$\sum_{e \in E : W_i \in \rho(e)} \eta(e) \leq \frac{4}{t} \sum_{e \in E} \eta(e).$$

For pseudocode and a proof of this theorem, see Appendix C. We remark that when $t \geq n$, the algorithm can just construct a singleton set for every vertex.

For technical reasons, edges with stretch less than 1 can be inconvenient. So, we define

$$\eta(e) = \max(\mathrm{st}_T(e), 1) \quad \text{and} \quad \eta(E) = \sum_{e \in E} \eta(e). \tag{10.2}$$

The tree T should always be clear from context.

Given a (T, E)-decomposition, $(\{W_1, \ldots, W_h\}, \rho)$, we define the map

$$\sigma : \{1, \ldots, h\} \times \{1, \ldots, h\} \to E \cup \{\text{undefined}\}$$

by setting

$$\sigma(i,j) = \begin{cases} \arg\max_{e : \rho(e) = \{W_i, W_j\}} \mathrm{weight}(e)/\eta(e) & \text{if } i \neq j \text{ and such an } e \text{ exists,} \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{10.3}$$

In the event of a tie, we let e be the lexicographically least edge maximizing $\mathrm{weight}(e)/\eta(e)$ such that $\rho(e) = \{W_i, W_j\}$. Note that $\sigma(i,j)$ is a weighted edge.

The map σ tells us which edge from E between W_i and W_j to add to T. The following property of σ, which follows immediately from its definition, will be used in our analysis in this and the next section.

PROPOSITION 10.4. *For every i, j such that $\sigma(i,j)$ is defined and for every $e \in E$ such that $\rho(e) = \{W_i, W_j\}$,*

$$\frac{\mathit{weight}(e)}{\eta(e)} \leq \frac{\mathit{weight}(\sigma(i,j))}{\eta(\sigma(i,j))}.$$

We can now state the procedure by which we augment a spanning tree.

$F = \texttt{AugmentTree}(T, E, t)$,
E is set of weighted edges,
T is a spanning tree of the vertices underlying E,
t is an integer.
1. Compute $\mathrm{st}_T(e)$ for each edge $e \in E$.
2. $((W_1, \ldots, W_h), \rho) = \texttt{decompose}(T, E, \eta, t)$, where $\eta(e)$ is as defined in (10.2).
3. Set F to be the union of the weighted edges $\sigma(i,j)$ over all pairs $1 \le i < j \le h$ for which $\sigma(i,j)$ is defined, where $\sigma(i,j)$ is as defined in (10.3).

$A = \texttt{UltraSimple}(E, t)$
1. Set $T = \texttt{LowStretch}(E)$.
2. Set $F = \texttt{AugmentTree}(T, E, t)$.
3. Set $A = T \cup F$.

We remark that when $t \ge n$, `UltraSimple` can just return $A = E$.

THEOREM 10.5 (`AugmentTree`). *On input a set of weighted edges E, a spanning subtree T, and an integer $1 < t \le \eta(E)$, the algorithm* ***`AugmentTree`*** *runs in time $O(m \log n)$, where $m = |E|$. The set of edges F output by the algorithm satisfies the following:*

(a) $F \subseteq E$.
(b) $|F| \le t^2/2$.
(c) *If $T \subseteq E$, as happens when* ***`AugmentTree`*** *is called by* ***`UltraSimple`****, then $(T \cup F) \preccurlyeq E$.*
(d)

$$E \preccurlyeq \frac{12\eta(E)}{t} \cdot (T \cup F). \tag{10.4}$$

Moreover, if E is planar, then A is planar and $|F| \le 3t - 6$.

Proof. In Appendix B, we present an algorithm for computing the stretch of each edge of E in time $O(m \log n)$. The remainder of the analysis of the running time is trivial. Part (a) follows immediately from the statement of the algorithm. When $T \subseteq E$, $T \cup F \subseteq E$, so part (c) follows as well.

To verify (b), note that the algorithm adds at most one edge to F for each pair of sets in $W_1, \ldots, W_h$, and there are at most $\binom{t}{2} \le t^2/2$ such pairs. If E is planar, then F must be planar as F is a subgraph of E. Moreover, we can use Lemma C.1 to show that the graph induced by E on the sets $W_1, \ldots, W_h$ is also planar. Thus, the number of pairs of these sets connected by edges of E is at most the maximum number of edges in a planar graph with t vertices, $3t - 6$.

We now turn to the proof of part (d). Set

$$\beta = 4\eta(E)/t. \tag{10.5}$$

By Theorem 10.3, ρ and $W_1, \ldots, W_h$ satisfy

$$\sum_{e : W_i \in \rho(e)} \eta(e) \le \beta \quad \text{for all } W_i \text{ such that } |W_i| > 1. \tag{10.6}$$

Let E_i^{int} denote the set of edges e with $\rho(e) = \{W_i\}$, and let E_i^{ext} denote the set of edges e with $|\rho(e)| = 2$ and $W_i \in \rho(e)$. Let $E^{int} = \cup_i E_i^{int}$ and $E^{ext} = \cup_i E_i^{ext}$. Also, let T_i denote the tree formed by the edges of T inside the set W_i. Note that when $|W_i| = 1$, T_i and E_i^{int} are empty.

We will begin by proving that when $|W_i| > 1$,

$$E_i^{int} \preccurlyeq \left(\sum_{e \in E_i^{int}} \eta(e) \right) T_i, \tag{10.7}$$

from which it follows that

$$E^{int} \preccurlyeq \sum_{i:|W_i|>1} \left(\sum_{e \in E_i^{int}} \eta(e) \right) T_i. \tag{10.8}$$

For any edge $e \in E_i^{int}$, the path in T between the endpoints of e lies entirely in T_i. So, by (9.2) we have

$$e \preccurlyeq \mathrm{st}_T(e) \cdot T_i \preccurlyeq \eta(e) \cdot T_i.$$

Inequality (10.7) now follows by summing over the edges $e \in E_i^{int}$.

We now define the map $\tau : E \to E \cup \{\text{undefined}\}$ by

$$\tau(e) = \begin{cases} \sigma(i,j) & \text{if } |\rho(e)| = 2, \text{ where } \rho(e) = \{W_i, W_j\}, \text{ and} \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{10.9}$$

To handle the edges bridging components, we prove that for each edge e with $\rho(e) = (W_i, W_j)$,

$$e \preccurlyeq 3\eta(e)(T_i + T_j) + 3\frac{\mathrm{weight}(e)}{\mathrm{weight}(\tau(e))} \cdot \tau(e). \tag{10.10}$$

Let $e = w\big(u, v\big)$ be such an edge, with $u \in W_i$ and $v \in W_j$. Let $\tau(e) = z\big(x, y\big)$, with $x \in W_i$ and $y \in W_j$. Let t_i denote the last vertex in T_i on the path in T from u to v (see Figure 3). If T_i is empty, $t_i = u$. Note that t_i is also the last vertex in T_i on the

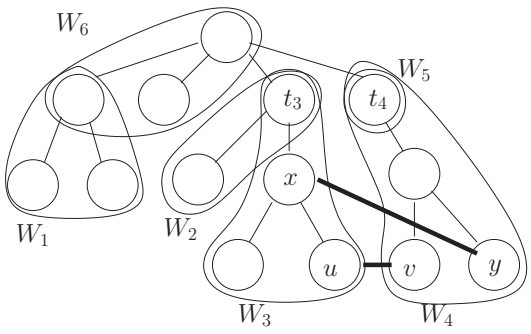

FIG. 3. *In this example,* $e = w\big(u, v\big)$ *and* $\tau(e) = z\big(x, y\big)$.

path in T from x to y. Define t_j similarly. As $T_i(u,x) \subseteq T_i(u,t_i) \cup T_i(t_i,x)$, the tree T_i contains a path from u to x of resistance at most

$$\text{resistance}(T_i(u,t_i)) + \text{resistance}(T_i(t_i,x)),$$

and the tree T_j contains a path from y to v of resistance at most

$$\text{resistance}(T_j(y,t_j)) + \text{resistance}(T_j(t_j,v)).$$

Furthermore, as $T_i(u,t_i) + T_j(t_j,v) \subseteq T(u,v)$ and $T_i(t_i,x) + T_j(y,t_j) \subseteq T(x,y)$, the sum of the resistances of the paths from u to x in T_i and from y to v in T_j is at most

$$\begin{aligned}\text{resistance}(T(u,v)) + \text{resistance}(T(x,y)) &= \text{st}_T(e)/w + \text{st}_T(\tau(e))/z \\ &\le \eta(e)/w + \eta(\tau(e))/z \\ &\le 2\eta(e)/w,\end{aligned}$$

where the last inequality follows from Proposition 10.4. Thus, the graph

$$3\eta(e)(T_i + T_j) + 3w\big(x,y\big) = 3\eta(e)(T_i + T_j) + 3\frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e)$$

contains a path from u to v of resistance at most

$$\frac{2}{3}\frac{1}{w} + \frac{1}{3}\frac{1}{w} = \frac{1}{w},$$

which by Lemma 9.1 implies (10.10).

We will now sum (10.10) over every edge $e \in E_i^{ext}$ for every i, observing that this counts every edge in E^{ext} twice.

$$\begin{aligned}E^{ext} &= (1/2)\sum_i \sum_{e \in E_i^{ext}} e \\ &\preccurlyeq \sum_i \sum_{e \in E_i^{ext}} 3\eta(e)T_i + (1/2)\sum_i \sum_{e \in E_i^{ext}} 3\frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) \\ &= 3\sum_i \left(\sum_{e \in E_i^{ext}} \eta(e) \right) T_i + 3 \sum_{e \in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) \\ &= 3 \sum_{i:|W_i|>1} \left(\sum_{e \in E_i^{ext}} \eta(e) \right) T_i + 3 \sum_{e \in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e),\end{aligned} \tag{10.11}$$

as T_i is empty when $|W_i| = 1$.

We will now upper bound the right-hand side of (10.11). To handle boundary cases, we divide E^{ext} into two sets. We let E^{ext}_{single} consist of those $e \in E^{ext}$ for which both sets in $\rho(e)$ have size 1. We let $E^{ext}_{general} = E^{ext} - E^{ext}_{single}$ contain the rest of the edges in E^{ext}. For $e \in E^{ext}_{single}$, $\tau(e) = e$, while for $e \in E^{ext}_{general}$, $\tau(e) \in E^{ext}_{general}$.

For E^{ext}_{single}, we have

$$\sum_{e \in E^{ext}_{single}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) = \sum_{e \in E^{ext}_{single}} \tau(e) = E^{ext}_{single}.$$

To evaluate the sum over the edges $e \in E^{ext}_{general}$, consider any $f \in E^{ext}_{general}$ in the image of τ. Let i be such that $f \in E^{ext}_i$ and $|W_i| > 1$. Then, for every e such that $\tau(e) = f$, we have $e \in E^{ext}_i$. So, by Proposition 10.4,

(10.12)

$$\sum_{\substack{e \in E^{ext} \\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} = \sum_{\substack{e \in E^{ext}_i \\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}$$

$$\leq \sum_{e \in E^{ext}_i} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \leq \sum_{e \in E^{ext}_i} \frac{\eta(e)}{\eta(\tau(e))} \leq \sum_{e \in E^{ext}_i} \eta(e) \leq \beta.$$

Thus,

$$\sum_{e \in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) \preccurlyeq E^{ext}_{single} + \sum_{\substack{f \in \text{image}(\tau) \\ f \in E^{ext}_{general}}} \beta \cdot f \preccurlyeq \beta \cdot F.$$

Plugging this last inequality into (10.11), we obtain

$$E^{ext} \preccurlyeq 3 \sum_{i:|W_i|>1} \left(\sum_{e \in E^{ext}_i} \eta(e) \right) T_i + 3\beta \cdot F.$$

Applying (10.8) and then (10.6), we compute

$$E = E^{ext} + E^{int} \preccurlyeq 3 \sum_{i:|W_i|>1} T_i \left(\sum_{e \in E^{int}_i} \eta(e) + \sum_{e \in E^{ext}_i} \eta(e) \right) + 3\beta \cdot F \preccurlyeq 3\beta \cdot (T \cup F),$$

which by (10.5) implies the lemma. □

We now discuss a source of slack in Theorem 10.5. This is the motivation for the construction of ultra-sparsifiers in the next section.

In the proof of Theorem 10.5, we assume in the worst case that the tree decomposition could result in each tree T_i being connected to $t-1$ other trees, for a total of $t(t-1)/2$ extra edges. Most of these edges seem barely necessary, as they could be included at a small fraction of their original weight. To see why, consider the crude estimate at the end of inequality (10.12). We upper bound the multiplier of one bridge edge f from T_i,

$$\sum_{\substack{e \in E^{ext}_i \\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))},$$

by the sum of the multipliers of all bridge edges from T_i,

$$\sum_{e \in E^{ext}_i} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}.$$

The extent to which this upper bound is loose is the factor by which we could decrease the weight of the edge f in the preconditioner.

While we do not know how to accelerate our algorithms by decreasing the weights with which we include edges, we are able to use sparsifiers to trade many low-weight edges for a few edges of higher weight. This is how we reduce the number of edges we add to the spanning tree to $O(t \log^{c_2+5} n)$.

11. Ultra-sparsifiers. We begin our construction of ultra-sparsifiers by building ultra-sparsifiers for the special case in which our graph has a distinguished vertex r and a low-stretch spanning tree T with the property that for every edge $e \in E - T$, the path in T connecting the endpoints of e goes through r. In this case, we will call r the *root* of the tree. All the complications of ultra-sparsification will be handled in this construction. The general construction will follow simply by using tree splitters to choose the roots and decompose the input graph.

$E_s = \texttt{RootedUltraSparsify}(E, T, r, t, p)$

Condition: for all $e \in E$, $r \in T(e)$. The parameter t is a positive integer at most $\lceil \eta(E) \rceil$.

1. Compute $\mathrm{st}_T(e)$ and $\eta(e)$ for each edge $e \in E$, where η is as defined in (10.2).
2. If $t \geq |E|$, return $E_s = E$.
3. Set $(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t)$.
4. Compute σ, as given by (10.3), everywhere it is defined.
5. For every (i, j) such that $\sigma(i, j)$ is defined, set
   (11.1)
   $$\omega(i,j) = \sum_{e \in E : \rho(e) = \{W_i, W_j\}} \mathrm{weight}(e) \quad \text{and} \quad \psi(i,j) = \omega(i,j)/\mathrm{weight}(\sigma(i,j)).$$
6. Set $F = \{\psi(i,j)\sigma(i,j) : \sigma(i,j) \text{ is defined}\}$.
7. For each $f = \psi(i,j)\sigma(i,j) \in F$, set
   (11.2)
   $$\phi(f) = \max(\psi(i,j), \mathrm{st}_T(f)).$$
8. For $b \in \{1, \ldots, \lceil \log_2 \eta(E) \rceil\}$:
   (a) Set $F^b = \begin{cases} \{f \in F : \phi(f) \in [1,2]\} & \text{if } b = 1, \\ \left\{f \in F : \phi(f) \in (2^{b-1}, 2^b]\right\} & \text{otherwise.} \end{cases}$
   (b) Let H^b be the set of edges on vertex set $\{1, \ldots, h\}$ defined by
   $$H^b = \left\{\omega(i,j)\mathbf{(}i,j\mathbf{)} : \psi(i,j)\sigma(i,j) \in F^b\right\}.$$
   (c) Set $H^b_s = \texttt{Sparsify2}(H^b, p)$.
   (d) Set
   $$E^b_s = \left\{\sigma(i,j) : \exists w \text{ such that } w\mathbf{(}i,j\mathbf{)} \in H^b_s\right\}.$$
9. Set $E_s = \cup_b E^b_s$.

The algorithm `RootedUltraSparsify` begins by computing the same set of edges $\sigma(i, j)$ as was computed by `UltraSimple`. However, when `RootedUltraSparsify` puts one of these edges into the set F, it gives it a different weight: $\omega(i, j)$. For technical reasons, the set F is decomposed into subsets F^b according to the quantities $\phi(f)$, which will play a role in the analysis of `RootedUltraSparsify` analogous to the role played by $\eta(e)$ in the analysis of `UltraSimple`. Each set of edges F^b is sparsified, and the union of the edges of E that appear in the resulting sparsifiers is returned by the algorithm. The edges in F^b cannot necessarily be sparsified directly, as they might

all have different endpoints. Instead, F^b is first projected to a graph H^b on vertex set $\{1,\ldots,h\}$. After a sparsifier H^b_s of H^b is computed, it is lifted back to the original graph to form E^b_s. Note that the graph E_s returned by `RootedUltraSparsify` is a subgraph of E with the same edge weights.

We now prove that $F = \cup_{b=1}^{\lceil \log_2 \eta(E)\rceil} F^b$. Our proof will use the function η, which we recall was defined in (10.2) and which was used to define the map σ.

LEMMA 11.1. *For ϕ as defined in* (11.2)*, for every $f = \psi(i,j)\sigma(i,j) \in F$,*

$$1 \le \psi(i,j) \le \phi(f) \le \eta(E). \tag{11.3}$$

Proof. Recall from the definitions of ϕ and ψ that

$$\phi(f) \ge \psi(i,j) = \frac{\sum_{e\in E:\rho(e)=\{W_i,W_j\}} \text{weight}(e)}{\text{weight}(\sigma(i,j))}.$$

By definition $\sigma(i,j)$ is an edge in E satisfying $\rho(\sigma(i,j)) = \{W_i, W_j\}$; so, the right-hand side of the last expression is at least 1.

To prove the upper bound on $\phi(f)$, first apply Proposition 10.4 to show that

$$\psi(i,j) = \frac{\sum_{e\in E:\rho(e)=\{W_i,W_j\}} \text{weight}(e)}{\text{weight}(\sigma(i,j))} \le \frac{\sum_{e\in E:\rho(e)=\{W_i,W_j\}} \eta(e)}{\eta(\sigma(i,j))} \le \eta(E),$$

as η is always at least 1. Similarly,

$$\begin{aligned}\text{st}_T(f) &= \frac{\omega(i,j)}{\text{weight}(\sigma(i,j))}\text{st}_T(\sigma(i,j)) = \frac{\text{st}_T(\sigma(i,j))}{\text{weight}(\sigma(i,j))}\left(\sum_{e\in E:\rho(e)=\{W_i,W_j\}} \text{weight}(e)\right)\\ &\le \frac{\eta(\sigma(i,j))}{\text{weight}(\sigma(i,j))}\left(\sum_{e\in E:\rho(e)=\{W_i,W_j\}} \text{weight}(e)\right) \le \sum_{e\in E:\rho(e)=\{W_i,W_j\}} \eta(e) \le \eta(E),\end{aligned}$$

where the second-to-last inequality follows from Proposition 10.4. ∎

It will be convenient for us to extend the domain of ρ to F by setting $\rho(f) = \rho(e)$, where $e \in E$ has the same vertices as f, that is, when there exists $\gamma \in \mathbb{R}^+$ such that $f = \gamma e$. Define

$$\beta = 4\eta(E)/t.$$

Our analysis of `RootedUltraSparsify` will exploit the inequalities contained in the following two lemmas.

LEMMA 11.2. *For every i for which $|W_i| > 1$,*

$$\sum_{f\in F:W_i\in\rho(f)} st_T(f) \le \beta.$$

Proof. Consider any $f \in F$, and let $f = \psi(i,j)\sigma(i,j)$. Note that the weight of f is $\omega(i,j)$, and recall that $\text{st}_T(f) \le \eta(f)$. We first show that

$$\sum_{e:\tau(e)=\sigma(i,j)} \eta(e) \ge \eta(f).$$

By Proposition 10.4, and the definition of τ in (10.9)

$$\begin{aligned}
\sum_{e:\tau(e)=\sigma(i,j)} \eta(e) &\geq \frac{\eta(\sigma(i,j))}{\text{weight}(\sigma(i,j))} \sum_{e:\tau(e)=\sigma(i,j)} \text{weight}(e) \\
&= \frac{\eta(\sigma(i,j))}{\text{weight}(\sigma(i,j))} \text{weight}(f) \\
&= \max\left(\frac{\text{weight}(f)}{\text{weight}(\sigma(i,j))}, \frac{\text{st}_T(\sigma(i,j))}{\text{weight}(\sigma(i,j))} \text{weight}(f)\right) \\
&= \max\left(\psi(i,j), \text{st}_T(f)\right) \\
&= \max\left(\phi(f), \text{st}_T(f)\right) \quad \text{(by (11.2))} \\
&\geq \max\left(1, \text{st}_T(f)\right) \quad \text{(by (11.3))} \\
&= \eta(f).
\end{aligned}$$

We then have

$$\sum_{e\in E:W_i\in\rho(e)} \eta(e) \geq \sum_{f\in F:W_i\in\rho(f)} \eta(f).$$

The lemma now follows from the upper bound of $4\eta(E)/t$ imposed on the left-hand term by Theorem 10.3. □

LEMMA 11.3. *For every i for which $|W_i| > 1$,*

$$\sum_{f\in F:W_i\in\rho(f)} \phi(f) \leq 2\beta. \tag{11.4}$$

Proof. For an edge $f \in F$, let $\psi(f)$ equal $\psi(i,j)$, where $f = \psi(i,j)\sigma(i,j)$. With this notation, we may compute

$$\begin{aligned}
\sum_{f\in F:W_i\in\rho(f)} \phi(f) &\leq \sum_{f\in F:W_i\in\rho(f)} \text{st}_T(f) + \sum_{f\in F:W_i\in\rho(f)} \psi(f) \\
&\leq \sum_{f\in F:W_i\in\rho(f)} \eta(f) + \sum_{f\in F:W_i\in\rho(f)} \psi(f) \\
&\leq \beta + \sum_{f\in F:W_i\in\rho(f)} \psi(f)
\end{aligned}$$

by Lemma 11.2. We now bound the right-hand term as in the proof of inequality (10.12):

$$\sum_{f\in F:W_i\in\rho(f)} \psi(f) = \sum_{e\in E_i^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \leq \sum_{e\in E_i^{ext}} \frac{\eta(e)}{\eta(\tau(e))} \leq \sum_{e\in E_i^{ext}} \eta(e) \leq \beta$$

by our choice of β and Theorem 10.3. □

LEMMA 11.4 (`RootedUltraSparsify`). *Let T be a spanning tree on a vertex set V, and let E be a nonempty set of edges on V for which there exists an $r \in V$ be such that for all $e \in E$, $r \in T(e)$. For $p > 0$ and t a positive integer at most $\lceil \eta(E) \rceil$, let E_s be the graph returned by* `RootedUltraSparsify`*(E, T, r, t, p). The graph E_s is a subgraph of E, and with probability at least $1 - \lceil \log_2 \eta(E) \rceil\, p$,*

$$|E_s| \leq c_1 \log^{c_2}(n/p) \max(1, \lceil \log_2 \eta(E) \rceil) t \tag{11.5}$$

and

$$E \preccurlyeq (3\beta + 126\beta \max(1, \log_2 \eta(E))) \cdot T + 120\beta \cdot E_s, \tag{11.6}$$

where $\beta = 4\eta(E)/t$.

Proof. We first dispense with the case in which the algorithm terminates at line 2. If $t \geq m$, then both (11.5) and (11.6) are trivially satisfied by setting $E_s = E$, as $\beta \geq 2$.

By Theorem 1.3 each graph H_s^b computed by `Sparsify2` is a $c_1 \log^{c_2}(n/p)$-sparsifier of H^b according to Definition 1.2 with probability at least $1 - p$. As there are at most $\lceil \log_2 \eta(E) \rceil$ such graphs H^b, this happens for all these graphs with probability at least $1 - \lceil \log_2 \eta(E) \rceil p$. For the remainder of the proof, we will assume that each graph H_s^b is a $c_1 \log^{c_2}(n/p)$-sparsifier of H^b. Recalling that $h \leq t$, the bound on the number of edges in E_s is immediate.

Our proof of (11.6) will go through an analysis of intermediate graphs. As some of these could be multigraphs, we will find it convenient to write them as sums of edges.

To define these intermediate graphs, let r_i be the vertex in W_i that is closest to r in T. As in section 10, let T_i denote the edges of the subtree of T with vertex set W_i. We will view r_i as the root of tree T_i. Note that if $|W_i| = 1$, then $W_i = \{r_i\}$ and T_i is empty. As distinct sets W_i and W_j can overlap in at most one vertex, $\sum_i T_i \leq T$. We will exploit the fact that for each $e \in E$ with $\rho(e) = \{W_i, W_j\}$, the path $T(e)$ contains both r_i and r_j, which follows from the condition $r \in T(e)$.

We now define the edge set D^b, which is a projection of H^b to the vertex set $r_1, \ldots, r_h$, and D_s^b, which is an analogous projection of the sparsifier H_s^b. We set

$$D^b = \sum_{(i,j):\psi(i,j)\sigma(i,j)\in F^b} \omega(i,j)\big(r_i, r_j\big)$$

and

$$D_s^b = \sum_{w\big(i,j\big)\in H_s^b} w\big(r_i, r_j\big).$$

As the sets W_i and W_j are allowed to overlap slightly, it could be the case that some $r_i = r_j$ for $i \neq j$. In this case, D^b would not be isomorphic to H^b.

Set

$$F_s^b = \left\{\gamma\psi(i,j)\sigma(i,j) : \exists\gamma \text{ and } (i,j) \text{ so that } \gamma\omega(i,j)\big(i,j\big) \in H_s^b\right\}.$$

The edge set H^b can be viewed as a projection of the edge set F^b to the vertex set $\{1, \ldots, h\}$, and the edge set F_s^b can be viewed as a lift of H_s^b back into a reweighted subgraph of F^b.

We will prove the following inequalities:

$$E \preccurlyeq 3\beta \cdot T + 3F, \tag{11.7}$$

$$F^b \preccurlyeq 2\beta \cdot T + 2D^b, \tag{11.8}$$

$$D^b \preccurlyeq (5/4)D_s^b, \tag{11.9}$$

$$D_s^b \preccurlyeq 16\beta \cdot T + 2F_s^b, \tag{11.10}$$

$$F_s^b \preccurlyeq 8\beta \cdot E_s^b. \tag{11.11}$$

Inequality (11.6) in the statement of the lemma follows from these inequalities and $F = \sum_b F^b$.

To prove inequality (11.7), we exploit the proof of Theorem 10.5. The edges F constructed in `RootedUltraSparsify` are the same as those chosen by `UltraSimple`, except that they are reweighted by the function ψ. If we follow the proof of inequality (10.4) in Theorem 10.5, but neglect to apply inequality (10.12), we obtain

$$E \preccurlyeq 3\beta \cdot T + 3 \sum_{e \in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) = 3\beta \cdot T + 3F.$$

To prove inequality (11.8), consider any edge $w\big(u,v\big) = f \in F^b$. Assume $\rho(f) = \{W_i, W_j\}$, $u \in W_i$, and $v \in W_j$. We will now show that

$$f \preccurlyeq 2\text{st}_T(f)(T_i + T_j) + 2w\big(r_i, r_j\big). \tag{11.12}$$

As the path from u to v in T contains both r_i and r_j,

$$\text{resistance}(T(u, r_i)) + \text{resistance}(T(r_j, v)) \leq \text{resistance}(T(u, v)) = \text{st}_T(f)/w.$$

Thus, the resistance of the path

$$2\text{st}_T(f)T(u, r_i) + 2w\big(r_i, r_j\big) + 2\text{st}_T(f)T(r_j, v)$$

is at most $1/w$, and so Lemma 9.1 implies that

$$f \preccurlyeq 2\text{st}_T(f)T(u, r_i) + 2w\big(r_i, r_j\big) + 2\text{st}_T(f)T(r_j, v),$$

which in turn implies (11.12). Summing (11.12) over all $f \in F^b$ yields

$$\begin{aligned}
F^b &\preccurlyeq 2\sum_i \left(\sum_{f \in F : W_i \in \rho(f)} \text{st}_T(f) \right) T_i + 2D^b \\
F^b &\preccurlyeq 2 \sum_{i : |W_i| > 1} \left(\sum_{f \in F : W_i \in \rho(f)} \text{st}_T(f) \right) T_i + 2D^b \quad \text{as } T_i \text{ is empty when } |W_i| = 1 \\
&\preccurlyeq 2\sum_i \beta \cdot T_i + 2D^b \quad \text{by Lemma 11.2} \\
&\preccurlyeq 2\beta \cdot T + 2D^b.
\end{aligned}$$

We now prove inequality (11.10), as it uses similar techniques. Let $f_s = w\big(u,v\big) \in F_s^b$. Then, there exist γ and (i,j) so that $\gamma\omega(i,j)\big(i,j\big) \in H_s^b$, $u \in W_i$, and $v \in W_j$. Set $\gamma(f_s)$ to be this multiplier γ. By part (c) of Definition 1.2, we must have $\omega(i,j)\big(i,j\big) \in H^b$ and $\psi(i,j)\sigma(i,j) \in F^b$. Let $f = \psi(i,j)\sigma(i,j)$. Note that $f_s = \gamma(f_s)f$. The sum of the resistances of the paths from r_i to u in T_i and from v to r_j in T_j is

$$\text{resistance}(T(r_i, u)) + \text{resistance}(T(v, r_j)) \leq \text{resistance}(T(u, v)) = \text{st}_T(f)/\omega(i,j),$$

as $\text{weight}(f) = \omega(i,j)$. Thus, the resistance of the path

$$2\text{st}_T(f)T(r_i, u) + 2f + 2\text{st}_T(f)T(v, r_j)$$

is at most $1/\omega(i,j)$, and so Lemma 9.1 implies that

$$\omega(i,j)\mathbf{(}r_i,r_j\mathbf{)} \preccurlyeq 2\mathrm{st}_T(f)(T_i+T_j)+2f$$

and

$$\begin{aligned}\gamma(f_s)\omega(i,j)\mathbf{(}r_i,r_j\mathbf{)} &\preccurlyeq 2\gamma(f_s)\mathrm{st}_T(f)(T_i+T_j)+2f_s \\ &\preccurlyeq 2\gamma(f_s)\phi(f)(T_i+T_j)+2f_s && \text{(by (11.2))} \\ &\preccurlyeq 2^{b+1}\gamma(f_s)(T_i+T_j)+2f_s && \text{(by } f\in F^b\text{)}.\end{aligned}$$

Summing this inequality over all $f_s \in F_s^b$, we obtain

$$D_s^b \preccurlyeq \sum_i \left(2^{b+1}\sum_{f_s\in F_s^b: W_i\in\rho(f_s)}\gamma(f_s)\right)T_i + 2F_s^b.$$

For all i such that $|W_i|>1$,

$$\begin{aligned}\sum_{f_s\in F_s^b: W_i\in\rho(f_s)}\gamma(f_s) &\le 2\left|\left\{f\in F^b : W_i\in\rho(f)\right\}\right| && \text{(part (d) of Definition 1.2)} \\ &\le 2\sum_{f\in F^b: W_i\in\rho(f)}\phi(f)/2^{b-1} \\ &\le 4\beta/2^{b-1} && \text{(by Lemma 11.3)} \\ &= \beta/2^{b-3}. && (11.13)\end{aligned}$$

So,

$$D_s^b \preccurlyeq \sum_i 16\beta\cdot T_i + 2F_s^b \preccurlyeq 16\beta\cdot T + 2F_s^b.$$

To prove inequality (11.11), let f_s be any edge in F_s, let f be the edge in F such that $f_s=\gamma(f_s)f$, and let $\sigma(i,j)$ be the edge such that $f_s=\gamma(f_s)\psi(i,j)\sigma(i,j)$. It suffices to show that

$$\text{weight}(f_s) \le 8\beta\ \text{weight}(\sigma(i,j)). \tag{11.14}$$

Set b so that $f\in F^b$. By (11.13),

$$\gamma(f_s)\le \beta/2^{b-3}\le 8\beta/\phi(f) = 8\beta/\max(\psi(i,j),\mathrm{st}_T(f)) \le 8\beta/\psi(i,j).$$

As $\text{weight}(f_s)=\gamma(f_s)\psi(i,j)\text{weight}(\sigma(i,j))$, inequality (11.14) follows.

It remains to prove inequality (11.9). The only reason this inequality is not immediate from part (a) of Definition 1.2 is that we may have $r_i=r_j$ for some $i\neq j$. Let $R=\{r_1,\ldots,r_h\}$ and $S=\{1,\ldots,h\}$. Define the map $\pi:\mathbb{R}^R\to\mathbb{R}^S$ by $\pi(x)_i=x_{r_i}$. We then have for all $x\in\mathbb{R}^R$

$$x^T L_{D^b}x = \pi(x)^T L_{H^b}\pi(x) \quad\text{and}\quad x^T L_{D_s^b}x = \pi(x)^T L_{H_s^b}\pi(x);$$

so,

$$x^T L_{D^b}x = \pi(x)^T L_{H^b}\pi(x) \le (5/4)\pi(x)^T L_{H_s^b}\pi(x) = (5/4)x^T L_{D_s^b}x. \qquad \square$$

The algorithm `UltraSparsify` will construct a low-stretch spanning tree T of a graph, choose a root vertex r, apply `RootedUltraSparsify` to sparsify all edges whose path in T contains r, and then work recursively on the trees obtained by removing the root vertex from T. The root vertex will be chosen to be a tree *splitter*, where we recall that a vertex r is a splitter of a tree T if the trees $T^1, \ldots, T^q$ obtained by removing r each have at most half as many vertices as T. The existence of such a vertex was established by Jordan [34], and it is well-known that a tree splitter can be found in linear time. By making the root a splitter of the tree, we bound the depth of the recursion. This is critical both for bounding the running time of the algorithm and for proving a bound on the quality of the approximation it returns. For each edge e such that $r \notin T(e)$, $T(e)$ is entirely contained in one of $T^1, \ldots, T^q$. Such edges are sparsified recursively.

$U = \texttt{UltraSparsify}(G = (V, E), k)$
Condition: G is connected.

1. $T = \texttt{LowStretch}(E)$.
2. Set $t = 517 \cdot \max(1, \log_2 \eta(E)) \cdot \lceil \log_2 n \rceil \, \eta(E)/k$ and $p = \left(2 \lceil \log \eta(E) \rceil \, n^2\right)^{-1}$.
3. If $t \geq \eta(E)$, then set $A = E - T$; otherwise, set $A = \texttt{TreeUltraSparsify}(E - T, t, T, p)$.
4. $U = T \cup A$.

$A = \texttt{TreeUltraSparsify}(E', t', T', p)$

1. If $E' = \emptyset$, return $A = \emptyset$.
2. Compute a splitter r of T'.
3. Set $E^r = \{\text{edges } e \in E' \text{ such that } r \in T'(e)\}$ and $t_r = \lceil t' \eta(E^r)/\eta(E') \rceil$.
4. If $t_r > 1$, set $A^r = \texttt{RootedUltraSparsify}(E^r, T', r, t_r, p)$; otherwise, set $A^r = \emptyset$.
5. Set $T^1, \ldots, T^q$ to be the trees obtained by removing r from T'. Set $V^1, \ldots, V^q$ to be the vertex sets of these trees, and set $E^1, \ldots, E^q$ so that $E^i = \left\{(u, v) \in E' : \{u, v\} \subseteq V^i\right\}$.
6. For $i = 1, \ldots, q$, set
$$A = A^r \cup \texttt{TreeUltraSparsify}(E^i, t' \eta(E^i)/\eta(E'), T^i, p).$$

THEOREM 11.5 (ultra-sparsification). *On input a weighted, connected n-vertex graph $G = (V, E)$ and $k \geq 1$, `UltraSparsify`(E, k) returns a set of edges $U = T \cup A \subseteq E$ such that T is a spanning tree of G, $U \subseteq E$, and with probability at least $1 - 1/2n$,*

$$U \preccurlyeq E \preccurlyeq kU, \tag{11.15}$$

and

$$|A| \leq O\left(\frac{m}{k} \log^{c_2+5} n\right), \tag{11.16}$$

where $m = |E|$. Furthermore, `UltraSparsify` *runs in expected time $O(m \log^c n)$, for some constant c.*

We remark that this theorem is very loose when $m/k \geq n$. In this case, the calls made to `decompose` by `RootedUltraSparsify` could have $t \geq n$, in which case `decompose` will just return singleton sets, and the output of `RootedUltraSparsify`

will essentially just be the output of `Sparsify2` on E^r. In this case, the upper bound in (11.16) can be very loose.

Proof. We first dispense with the case $t \geq \eta(E)$. In this case, `UltraSparsify` simply returns the graph E, so (11.15) is trivially satisfied. The inequality $t \geq \eta(E)$ implies $k \leq O(\log^2 n)$, so (11.16) is trivially satisfied as well.

At the end of the proof, we will use the inequality $t < \eta(E)$. It will be useful to observe that every time `TreeUltraSparsify` is invoked,

$$t' = t\eta(E')/\eta(E).$$

To apply the analysis of `RootedUltraSparsify`, we must have

$$t^r \leq \lceil \eta(E^r) \rceil .$$

This follows from

$$t^r = \lceil t'\eta(E^r)/\eta(E') \rceil = \lceil t\eta(E^r)/\eta(E) \rceil \leq \lceil \eta(E^r) \rceil ,$$

as `TreeUltraSparsify` is called only if $t < \eta(E)$.

Each vertex of V can be a root in a call to `RootedUltraSparsify` at most once, so this subroutine is called at most n times during the execution of `UltraSparsify`. Thus by Lemma 11.4, with probability at least

$$1 - n \lceil \log_2 \eta(E) \rceil \, p = 1 - 1/2n,$$

every graph E_s returned by a call to `RootedUltraSparsify` satisfies (11.5) and (11.6). Accordingly, we will assume both of these conditions hold for the rest of our analysis.

We now prove the upper bound on the number of edges in A. During the execution of `UltraSparsify`, many vertices become the root of some tree. For those vertices v that do not, set $t_v = 0$. By (11.5),

$$|A| = \sum_{r \in V : t_r > 1} |A^r| \leq c_1 \log^{c_2}(n/p) \max(1, \lceil \log_2 \eta(E) \rceil) \sum_{r \in V : t_r > 1} t_r. \tag{11.17}$$

As $\lceil z \rceil \leq 2z$ for $z \geq 1$ and $E^{r_1} \cap E^{r_2} = \emptyset$ for each $r_1 \neq r_2$,

$$\sum_{r \in V : t_r > 1} t_r = \sum_{r \in V : t_r > 1} \left\lceil \frac{\eta(E^r)}{\eta(E)} t \right\rceil \leq \sum_{r \in V : t_r > 1} \frac{2\eta(E^r)}{\eta(E)} t \leq 2t.$$

Thus,

$$\begin{aligned} (11.17) &\leq 2c_1 \log^{c_2}(n/p) \lceil \log_2 \eta(E) \rceil \, t \\ &\leq 2c_1 \log^{c_2}(n/p) \lceil \log_2 \eta(E) \rceil \, 517 \cdot \log_2 \eta(E) \cdot \lceil \log_2 n \rceil \, \eta(E)/k \\ &\leq O\left(\frac{m}{k} \log^{c_2+5} n \right), \end{aligned}$$

where the last inequality uses $\eta(E) = O(m \log n \ \log\log n) = O(m \log^2 n)$ from Theorem 10.1 and $\log m = O(\log n)$.

We now establish (11.15). For every vertex r that is ever selected as a tree splitter in line 2 of `TreeUltraSparsify`, let T^r be the tree T' of which r is a splitter, and let E^r denote the set of edges and t_r be the parameter set in line 3. Observe that $\cup_r E^r = E - T$. Let

$$\beta_r = 4\eta(E^r)/t_r,$$

and note this is the parameter used in the analysis of `RootedUltraSparsify` in Lemma 11.4. If $t_r > 1$, let A^r be the set of edges returned by the call to `RootedUltraSparsify`. By Lemma 11.4, `RootedUltraSparsify` returns a set of edges A^r satisfying

$$E^r \preccurlyeq (3\beta_r + 126\beta_r \max(1, \log_2 \eta(E^r))) \cdot T^r + 120\beta_r \cdot A^r. \tag{11.18}$$

On the other hand, if $t_r = 1$ and so $A^r = \emptyset$, then $\beta_r = 4\eta(E^r)$. We know that (11.18) is satisfied in this case because $E^r \preccurlyeq \eta(E^r)T^r$ (by (9.1)). If $t_r = 0$, then $E^r = \emptyset$ and (11.18) is trivially satisfied. As $t_r = \lceil t\eta(E^r)/\eta(E) \rceil$,

$$\beta_r \leq 4\eta(E)/t.$$

We conclude

$$\begin{aligned} E^r &\preccurlyeq 129\beta_r \max(1, \log_2 \eta(E^r)) \cdot T^r + 120\beta_r \cdot A^r \\ &\preccurlyeq 516(\eta(E)/t) \max(1, \log_2 \eta(E^r))T^r + 120(\eta(E)/t)A^r. \end{aligned}$$

Adding T, summing over all r, and remembering $\eta(E^r) \leq \eta(E)$, we obtain

$$T + (E - T) \preccurlyeq T + 516(\eta(E)/t) \max(1, \log_2 \eta(E)) \sum_r T^r + 120(\eta(E)/t)A.$$

As r is always chosen to be a splitter of the tree input to `TreeUltraSparsify`, the depth of the recursion is at most $\lceil \log_2 n \rceil$. Thus, no edge of T appears more than $\lceil \log_2 n \rceil$ times in the sum $\sum_r T^r$, and we may conclude

$$\begin{aligned} T + (E - T) &\preccurlyeq T + 516(\eta(E)/t) \max(1, \log_2 \eta(E)) \lceil \log_2 n \rceil T + 120(\eta(E)/t)A \\ &\preccurlyeq 517(\eta(E)/t) \max(1, \log_2 \eta(E)) \lceil \log_2 n \rceil T + 120(\eta(E)/t)A \\ &\preccurlyeq k(T + A) \\ &= kU, \end{aligned}$$

where the second inequality follows from $t \leq \eta(E)$, and the third inequality follows from the value chosen for t in line 2 of `UltraSparsify`.

To bound the expected running time of `UltraSparsify`, first observe that the call to `LowStretch` takes time $O(m \log n \ \log\log n)$. Then, note that the routine `TreeUltraSparsify` is recursive, the recursion has depth at most $O(\log n)$, and all the graphs being processed by `TreeUltraSparsify` at any level of the recursion are disjoint. The running time of `TreeUltraSparsify` is dominated by the calls made to `Sparsify2` inside `RootedUltraSparsify`. Each of these takes nearly linear expected time, so the overall expected running time of `TreeUltraSparsify` is $O(m \log^c n)$ for some constant c. ∎

Appendix A. Gremban's reduction. Gremban [30] (see also [44]) provides the following method for handling positive off-diagonal entries. If A is an SDD_0-matrix, then Gremban decomposes A into $D + A_n + A_p$, where D is the diagonal of A, A_n is the matrix containing all the negative off-diagonal entries of A, and A_p contains all the positive off-diagonals. Gremban then considers the linear system

$$\widehat{A} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \hat{\boldsymbol{b}}, \qquad \text{where} \qquad \widehat{A} = \begin{bmatrix} D + A_n & -A_p \\ -A_p & D + A_n \end{bmatrix} \quad \text{and} \quad \hat{\boldsymbol{b}} = \begin{pmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{pmatrix},$$

and observes that $\boldsymbol{x} = (\boldsymbol{x}_1 - \boldsymbol{x}_2)/2$ will be the solution to $A\boldsymbol{x} = \boldsymbol{b}$, if a solution exists. Moreover, approximate solutions of Gremban's system yield approximate solutions of the original:

$$\left\| \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} - \widehat{A}^\dagger \hat{\boldsymbol{b}} \right\| \leq \epsilon \|\widehat{A}^\dagger \hat{\boldsymbol{b}}\| \qquad \text{implies} \qquad \left\| \boldsymbol{x} - A^\dagger \boldsymbol{b} \right\| \leq \epsilon \left\| A^\dagger \boldsymbol{b} \right\|,$$

where again $\boldsymbol{x} = (\boldsymbol{x}_1 - \boldsymbol{x}_2)/2$. Thus we may reduce the problem of solving a linear system in an SDD_0-matrix into that of solving a linear system in an SDDM_0-matrix that is at most twice as large and has at most twice as many nonzero entries.

Appendix B. Computing the stretch. We now show that given a weighted graph $G = (V, E)$ and a spanning tree T of G, we can compute $\mathrm{st}_T(e)$ for every edge $e \in E$ in $O((m+n)\log n)$ time, where $m = |E|$ and $n = |V|$.

For each pair of vertices $u, v \in V$, let resistance(u, v) be the resistance of $T(u, v)$, the path in T connecting u and v. We first observe that for an arbitrary $r \in V$, we can compute resistance(v, r) for all $v \in V$ in $O(n)$ time by a top-down traversal on the rooted tree obtained from T with root r. Using this information, we can compute the stretch of all edges in $E_r = \{\text{edges } e \in E \text{ such that } r \in T(e)\}$ in time $O(|E_r|)$.

We can then use tree splitters in the same manner as in `TreeUltraSparsify` to compute the stretch of all edges in E in $O((m+n)\log n)$ time. That is, in linear time we can identify a vertex r such that the removal of r from the tree produces subtrees, each of which has at most half as many vertices as the original. We then treat r as the root and compute the stretch of all edges in E_r. We note that these are exactly the edges whose endpoints are in different components of the forest obtained by removing all edges attached to vertex r. We then recurse on the subtrees obtained by removing the vertex r from the tree. This algorithm performs a linear amount of work on each level of the recursion, and our choice of splitters as roots of the trees guarantees that there are at most $\lceil \log_2 n \rceil$ levels of recursion.

Appendix C. Decomposing trees. The pseudocode for `decompose` appears on the next page. The algorithm performs a depth-first traversal of the tree, greedily forming sets W_j once they are attached to edges whose sum of η values exceeds a threshold ϕ. The traversal is performed by the subroutine `sub`. This routine is first called from the root of the tree, and it then recursively calls itself on the children of its input vertex before processing its input vertex. The routine `sub` returns a set of vertices, U, along with F, the set of edges touching vertices in U, and w, the sum of η over the edges in F. The vertices in U are those in the subtree rooted at v which `sub` did not place into a set W_j. While `sub` is gathering sets of vertices U_{sub}, the edges they are attached to are stored in F_{sub}, and the sum of the value of η on these edges is stored in w_{sub}.

There are four lines on which the routine `sub` can form a set W_j: line 3(c) ii, 6(b), 7(b), or 7(e). When the routine `sub` forms a set W_j of vertices, those vertices form a subtree of T. If the set is formed on line 7(e), then this subtree will be a singleton consisting of only one vertex. All the edges attached to those vertices, except possibly for the root of that subtree, are assigned to W_j by ρ. If the set W_j is formed in line 6(b) or 7(e), then all the edges attached to the root are assigned to W_j. Otherwise, the edges attached to the root are not assigned to W_j by ρ, unless they happen to also be attached to another vertex in W_j.

$(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t)$

Comment: h, ρ, and the W_i's are treated as global variables.

1. Set $h = 0$. *(h is incremented as sets are created)*
2. For all $e \in E$, set $\rho(e) = \emptyset$.
3. Set $\phi = 2\sum_{e \in E} \eta(e)/t$. *(the threshold above which sets are formed)*
4. $(F, w, U) = \texttt{sub}(r)$.
5. If $U \neq \emptyset$,
   (a) $h = h + 1$.
   (b) $W_h = U$.
   (c) For all $e \in F$, set $\rho(e) = \rho(e) \cup \{W_h\}$.

$(F, w, U) = \texttt{sub}(v)$

U is a set of vertices, F is the set of edges attached to U, and w is the sum of η over F

1. Let $v_1, \ldots, v_s$ be the children of v.
2. Set $w_{sub} = 0$, $F_{sub} = \emptyset$ and $U_{sub} = \emptyset$.
3. For $i = 1, \ldots, s$
   (a) $(F_i, w_i, U_i) = \texttt{sub}(v_i)$.
   (b) $w_{sub} = w_{sub} + w_i$, $F_{sub} = F_{sub} \cup F_i$, $U_{sub} = U_{sub} \cup U_i$.
   (c) If $w_{sub} \geq \phi$,
      i. $h = h + 1$.
      ii. Set $W_h = U_{sub} \cup \{v\}$.
      iii. For all $e \in F_{sub}$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
      iv. Set $w_{sub} = 0$, $F_{sub} = \emptyset$ and $U_{sub} = \emptyset$.
4. Set $F_v = \{(u, v) \in E\}$, the edges attached to v.
5. Set $w_v = \sum_{e \in F_v} \eta(e)$.
6. If $\phi \leq w_v + w_{sub} \leq 2\phi$,
   (a) $h = h + 1$.
   (b) Set $W_h = U_{sub} \cup \{v\}$.
   (c) For all $e \in F_{sub} \cup F_v$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (d) Return $(\emptyset, 0, \emptyset)$.
7. If $w_v + w_{sub} > 2\phi$,
   (a) $h = h + 1$.
   (b) Set $W_h = U_{sub} \cup \{v\}$.
   (c) For all $e \in F_{sub}$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (d) $h = h + 1$.
   (e) Set $W_h = \{v\}$. *(create a singleton set)*
   (f) For all $e \in F_v$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (g) Return $(\emptyset, 0, \emptyset)$.
8. Return $(F_{sub} \cup F_v, w_{sub} + w_v, U_{sub} \cup \{v\})$

There are three ways that `sub` could decide to create sets W_j. The first is if some subset of the children of v return sets F_i whose values under η sum to more than ϕ. In this case, `sub` collects the corresponding sets of vertices, along with v, into a set W_j. But, the edges attached to v do not necessarily get assigned by ρ to W_j as v was merely included in the set to make it a connected subtree. The second is if the sum of η over the edges attached to v and the sets U_i returned by a bunch of the

children exceeds ϕ but is less than 2ϕ. In this case, v and all those sets U_i are bundled together into a set W_j in line 6(b). Finally, if the sum of η over the edges attached to v and the sets U_i returned by a bunch of the children exceeds 2ϕ, then v is added as a singleton set, and a set is created containing the union of v with those sets U_i.

We assume that some vertex r has been chosen to be the root of the tree. This choice is used to determine which nodes in the tree are children of each other.

Proof of Theorem 10.3. As algorithm `decompose` traverses the tree T once and visits each edge in E once, it runs in linear time.

In our proof, we will say that an edge e is assigned to a set W_j if $W_j \in \rho(e)$. To prove part (a) of the theorem, we use the following observations: If W_j is formed in step 3(c) ii or step 6(b), then the sum of η over edges assigned to W_j is at least ϕ, and if W_j is formed in step 7(b), then the sum of η of edges incident to W_j and W_{j+1} (which is a singleton) is at least 2ϕ. Finally, if a set W_h is formed in line 5(b) of `decompose`, then the sum of η over edges assigned to W_h is greater than zero. But, at most one set is formed this way. As each edge is assigned to at most two sets in $W_1, \ldots, W_h$, we may conclude

$$2 \sum_{e \in E} \eta(e) > (h-1)\phi,$$

which implies $t > h - 1$. As both t and h are integers, this implies $t \geq h$.

We now prove part (b). First, observe that steps 6 and 7 guarantee that when a call to `sub`(v) returns a triple (F, w, U),

$$w = \sum_{e \in F} \eta(e) < \phi.$$

Thus, when a set W_h is formed in step 3(c) ii, we know that the sum of η over edges assigned to W_h equals w_{sub} and is at most 2ϕ. Similarly, we may reason that $w_{sub} < \phi$ at step 4. If a set W_h is formed in step 6(b), the sum of η over edges associated with W_h is $w_v + w_{sub}$ and must be at most 2ϕ. If a set W_h is formed in step 7(b), the sum of η over edges associated with W_h is w_{sub}, which we established is at most ϕ. As the set formed in step 7(e) is a singleton, we do not need to bound the sum of η over its associated edges. □

LEMMA C.1. *Suppose $G = (V, E)$ is a planar graph, π is a planar embedding of G, T is a spanning tree of G, and $t > 1$ is an integer. Let $(\{W_1, \ldots, W_h\}, \rho) =$ `decompose`(T, E, η, t) with the assumption that in step 1 of `sub`, the children $v_1, \ldots, v_s$ of v always appear in clockwise order according to π. Then the graph $G_{\{W_1,\ldots,W_h\}} = (\{1, \ldots, h\}, \{(i,j) : \exists\, e \in E, \rho(e) = \{W_i, W_j\}\})$ is planar.*

Proof. Recall that the contraction of an edge $e = (u, v)$ in a planar graph $G = (V, E)$ defines a new graph $(V - \{u\}, E \cup \{(x, v) : (x, u) \in E\} - \{(x, u) \in E\})$. Also recall that edge deletions and edge contractions preserve planarity.

We first prove the lemma in the special case in which the sets $W_1, \ldots, W_h$ are disjoint. For each j, let T_j be the graph induced on T by W_j. As each T_j is connected, $G_{\{W_1,\ldots,W_h\}}$ is a subgraph of the graph obtained by contracting all the edges in each subgraph T_j. Thus in this special case $G_{\{W_1,\ldots,W_h\}}$ is planar.

We now analyze the general case, recalling that the sets $W_1, \ldots, W_h$ can overlap. However, the only way sets W_j and W_k with $j < k$ can overlap is if the set W_j was formed at step 3(c) ii, and the vertex v becomes part of W_k after it is returned by a call to `sub`. In this situation, no edge is assigned to W_j for having v as an endpoint.

That is, the only edges of form (x, v) that can be assigned to W_j must have $x \in W_j$. So, these edges will not appear in $G_{\{W_1,\ldots,W_h\}}$.

Accordingly, for each j we define

$$X_j = \begin{cases} W_j - v & \text{if } W_j \text{ was formed at step 3(c) ii, and} \\ W_j & \text{otherwise.} \end{cases}$$

We have shown that $G_{\{W_1,\ldots,W_h\}} = G_{\{X_1,\ldots,X_h\}}$. Moreover, the sets $X_1, \ldots, X_h$ are disjoint. Our proof would now be finished if only each subgraph of G induced by a set X_j were connected. While this is not necessarily the case, we can make it the case by adding edges to E.

The only way the subgraph of G induced on a set X_j can fail to be connected is if W_j is formed at line 3(c) ii from the union of v with a collection sets U_i for $i_0 \le i \le i_1$ returned by recursive calls to `sub`. Now, consider what happens if we add edges of the form (v_i, v_{i+1}) to the graph for $i_0 \le i < i_1$, whenever they are not already present. As the vertices $v_{i_0}, \ldots, v_{i_1}$ appear in clockwise order around v, the addition of these edges preserves the planarity of the graph. Moreover, their addition makes the induced subgraphs on each set X_j connected, so we may conclude that $G_{\{X_1,\ldots,X_h\}}$ is in fact planar. ∎

REFERENCES

[1] 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, 2008, pp. 781–790.

[2] I. Abraham and O. Neiman, *Using petal-decompositions to build a low stretch spanning tree*, in Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC '12), 2012, pp. 395–406.

[3] N. Alon, R. M. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.

[4] W. N. Anderson and T. D. Morley, *Eigenvalues of the laplacian of a graph*, Linear Multilinear Algebra, 18 (1985), pp. 141–145.

[5] O. Axelsson, *A survey of preconditioned iterative methods for linear systems of algebraic equations*, BIT, 25 (1985), pp. 165–187.

[6] J. D. Batson, D. A. Spielman, and N. Srivastava, *Twice-Ramanujan sparsifiers*, in Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 2009, pp. 255–262.

[7] M. Bebendorf and W. Hackbusch, *Existence of H-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients*, Numer. Math., 95 (2003), pp. 1–28.

[8] M. Belkin, I. Matveeva, and P. Niyogi, *Regularization and semi-supervised learning on large graphs*, in Proceedings of the 17th Conference on Learning Theory, 2004, pp. 624–638.

[9] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, *Support-graph preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 930–951.

[10] E. Boman and B. Hendrickson, *On Spanning Tree Preconditioners*, manuscript, Sandia National Laboratories, 2001.

[11] E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo, *Maximum-weight-basis preconditioners*, Numer. Linear Algebra Appl., 11 (2004), pp. 695–721.

[12] E. G. Boman, D. Chen, O. Parekh, and S. Toledo, *On factor width and symmetric h-matrices*, Linear Algebra Appl., 405 (2005), pp. 239–248.

[13] E. G. Boman and B. Hendrickson, *Support theory for preconditioning*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 694–717.

[14] E. G. Boman, B. Hendrickson, and S. A. Vavasis, *Solving elliptic finite element systems in near-linear time with support preconditioners*, SIAM J. Numer. Anal., 46 (2008), pp. 3264–3284.

[15] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2001.

[16] D. Chen and S. Toledo, *Vaidya's preconditioners: Implementation and experimental study*, Electron. Trans. Numer. Anal., 16 (2003), pp. 30–49.

[17] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng, *Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, STOC '11, New York, 2011, pp. 273–282.
[18] D. Coppersmith and S. Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput., 11 (1982), pp. 472–492.
[19] S. I. Daitch and D. A. Spielman, *Support-Graph Preconditioners for* 2*-Dimensional Trusses*, CoRR abs/cs/0703119, 2007.
[20] S. I. Daitch and D. A. Spielman, *Faster approximate lossy generalized flow via interior point algorithms*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing, 2008, pp. 451–460.
[21] J. Demmel, *On Floating Point Errors in Cholesky*, Tech. report, Courant Institute, 1989.
[22] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Science Publications, New York, 1986.
[23] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, *Lower-stretch spanning trees*, SIAM J. Comput., 32 (2008), pp. 608–628.
[24] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Math. J., 23 (1973), pp. 298–305.
[25] A. Frangioni and C. Gentile, *New preconditioners for KKT systems of network flow problems*, SIAM J. Optim., 14 (2004), pp. 894–913.
[26] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[27] J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numer. Math., 50 (1987), pp. 377–404.
[28] G. H. Golub and M. Overton, *The convergence of inexact Chebychev and Richardson iterative methods for solving linear systems*, Numer. Math., 53 (1988), pp. 571–594.
[29] G. H. Golub and R. S. Varga, *Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods*, Numer. Math., 3 (1961), pp. 157–168.
[30] K. Gremban, *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1996.
[31] K. D. Gremban, G. L. Miller, and M. Zagha, *Performance evaluation of a new parallel preconditioner*, in Proceedings of the 9th International Symposium on Parallel Processing, IEEE Computer Society, 1995, pp. 65–69.
[32] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.
[33] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
[34] C. Jordan, *Sur les assemblages de lignes*, J. Reine Angew. Math., 70 (1869), pp. 185–190.
[35] A. Joshi, *Topics in Optimization and Sparse Linear Systems*, Ph.D. thesis, UIUC, 1997.
[36] J. A. Kelner and A. Madry, *Faster generation of random spanning trees*, in Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, 2009, pp. 13–21.
[37] J. A. Kelner, G. L. Miller, and R. Peng, *Faster approximate multicommodity flow using quadratically coupled flows*, in Proceedings of the 44th Symposium on Theory of Computing, STOC '12, New York, 2012, pp. 1–18.
[38] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, *A simple, combinatorial algorithm for solving sdd systems in nearly-linear time*, in Proceedings of the 45th Annual ACM Symposium on Theory of Computing, STOC '13, 2013, pp. 911–920.
[39] A. Kolla, Y. Makarychev, A. Saberi, and S.-H. Teng, *Subgraph sparsification and nearly optimal ultrasparsifiers*, in Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC '10, 2010, pp. 57–66.
[40] I. Koutis, G. L. Miller, and R. Peng, *Approaching optimality for solving sdd linear systems*, in Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 2010, pp. 235–244.
[41] I. Koutis, G. L. Miller, and R. Peng, *A nearly-mlogn time solver for sdd linear systems*, in Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, 2011, pp. 590–598.
[42] I. Koutis and G. 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, 2007, pp. 1002–1011.
[43] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[44] B. M. Maggs, G. L. Miller, O. Parekh, R. Ravi, and S. L. M. Woo, *Finding effective support-tree preconditioners*, in Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2005, pp. 176–185.

[45] A. N. Malyshev, *A note on the stablity of gauss-jordan elimination for diagonally dominant matrices*, Computing, 65 (2000), pp. 281–284.

[46] M. Mihail, *Conductance and convergence of Markov chains—A combinatorial treatment of expanders*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 526–531.

[47] J. Reif, *Efficient approximate solution of sparse linear systems*, Comput. Math. Appl., 36 (1998), pp. 37–58.

[48] A. Sankar, D. A. Spielman, and S.-H. Teng, *Smoothed analysis of the condition numbers and growth factors of matrices*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 446–476.

[49] G. Shklarski and S. Toledo, *Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 7–40.

[50] A. Sothers, *On the Complexity of Matrix Multiplication*, Ph.D. thesis, University of Edinburg, 2010.

[51] D. A. Spielman and N. Srivastava, *Graph sparsification by effective resistances*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing, 2008, pp. 563–568.

[52] D. A. Spielman and S.-H. Teng, *A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning*, SIAM J. Comput., 42 (2013), pp. 1–26.

[53] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 81–90; also available online from http://arxiv.org/abs/cs.DS/0310051.

[54] D. A. Spielman and S.-H. Teng, *Spectral sparsification of graphs*, SIAM J. Comput., 40 (2011), pp. 981–1025.

[55] D. A. Spielman and J. Woo, *A Note on Preconditioning by Low-Stretch Spanning Trees*, arXiv:0903.2816, 2009.

[56] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.

[57] S.-H. Teng, *The Laplacian paradigm: Emerging algorithms for massive graphs*, in Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation, TAMC'10, 2010, pp. 2–14.

[58] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[59] P. M. Vaidya, *Solving Linear Equations with Symmetric Diagonally Dominant Matrices by Constructing Good Preconditioners*, manuscript, UIUC, 1990.

[60] V. Vassilevska Williams, *Multiplying matrices faster than Coppersmith-Winograd*, in Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC '12), 2012, pp. 887–898.

[61] F. Zhang, ed., *The Schur Complement and Its Applications*, Numer. Methods Algorithms 4, Springer, New York, 2005.

[62] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, *Learning with local and global consistency*, Adv. Neural Inf. Process. Syst. 16, 2003, pp. 321–328.

[63] X. Zhu, Z. Ghahramani, and J. D. Lafferty, *Semi-supervised learning using Gaussian fields and harmonic functions*, in Proceedings of the 20th International Conference on Machine Learning, 2003.