COMBINATORIAL PRECONDITIONERS FOR SCALAR ELLIPTIC FINITE-ELEMENTS PROBLEMS

HAIM AVRON, DORON CHEN, GIL SHKLARSKI, AND SIVAN TOLEDO

Abstract. We present a new preconditioner for linear systems arising from finite-elements discretizations of scalar elliptic partial differential equations (PDE's). The solver splits the collection $\{K_e\}$ of element matrices into a subset $E(t)$ of matrices that are approximable by diagonally-dominant matrices and a subset of matrices that are not approximable. The approximable K_e's are approximated by diagonally-dominant matrices L_e's that are scaled and assembled to form a global diagonally-dominant matrix $L = \sum_{e \in E(t)} \alpha_e L_e$. A combinatorial graph algorithm approximates L by another diagonally-dominant matrix M that is easier to factor. The sparsification M is scaled and added to the inapproximable elements; the sum $\gamma M + \sum_{e \notin E(t)} K_e$ is factored and used as a preconditioner. When all the element matrices are approximable, which is often the case, the preconditioner is provably efficient. Experimental results show that on problems in which some of the K_e's are ill conditioned, our new preconditioner is more effective than an algebraic multigrid solver, than an incomplete-factorization preconditioner, and than a direct solver.

1. Introduction

We present a new class of combinatorial preconditioners for matrices that arise from finite-elements discretization of scalar elliptic partial differential equations. Our algorithms construct the preconditioners in several phases. First, we construct an approximation L_e to each element matrix K_e. These approximate element matrices are symmetric and diagonally dominant. We then split the elements into two sets: the set $E(t)$ in which L_e is a good approximation of K_e, and the rest. (t is a parameter that determines how good we require approximations to be.) We then scale and assemble the good element-by-element approximations to form $L = \sum_{e \in E(t)} \alpha_e L_e$. Next, we use a combinatorial graph algorithm to construct an easy-to-factor approximation M of L. Finally, we compute a scaling factor γ for M and add γM to the inapproximable elements, to form $\gamma M + \sum_{e \notin E(t)} K_e$. This is the matrix that we use to precondition the finite-element stiffness matrix $K = \sum_e K_e$. We factor it using a sparse Cholesky factorization algorithm.

Date: April 2007.

We now describe our approach in more details. Our algorithms apply to finite-element discretizations of the following class of problems. Find $u\colon \Omega \longrightarrow \mathbb{R}$ satisfying

$$\begin{aligned} \nabla \cdot (\theta(x)\nabla u) &= -f \qquad \text{on } \Omega \\ u &= u_0 \qquad \text{on } \Gamma_1 \ , \\ \theta \partial u/\partial n &= g \qquad \text{on } \Gamma_2 \ . \end{aligned} \tag{1}$$

The domain Ω is a bounded open set of $\mathbb{R}^d$ and Γ_1 and Γ_2 form a partition of the boundary of Ω. The *conductivity* θ is a spatially varying d-by-d symmetric positive definite matrix, f is a scalar *forcing function*, u_0 is a *Dirichlet boundary condition* and g is a *Neumann boundary condition*.

We assume that the discretization of (1) leads to an algebraic system of equations

$$Kx = b \ .$$

The matrix $K \in \mathbb{R}^{n\times n}$ is called a *stiffness matrix*, and it is a sum of *element matrices*, $K = \sum_{e\in E} K_e$. Each element matrix K_e corresponds to a subset of Ω called a *finite element*. The elements are disjoint except perhaps for their boundaries and their union is Ω. We assume that each element matrix K_e is symmetric, positive definite or positive semidefinite, and zero outside a small set of n_e rows and columns. In most cases n_e is uniformly bounded by a small integer (in our experiments n_e is 4 or 10). We denote the set of nonzero rows and columns of K_e by $\mathcal{N}_e$.

For the problem (1), all the element matrices are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$. This is one of two key aspects of our method: the method only works when element matrices are either nonsingular or are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

Our algorithms start by constructing a symmetric diagonally-dominant (SDD) approximation L_e to each element matrix K_e. A row or a column that is zero in K_e is also zero in L_e, so the L_e's are also very sparse. Our approximations are provably good: the spectral distance between K_e and L_e is within a $n_e^2/2$ factor of the best possible for an SDD approximation of K_e. Moreover, a recent result by Boman, Hendrickson and Vavasis [7] shows that under fairly general conditions on the discretization method, for each K_e there is some SDD matrix $\tilde{L}_e$ that is within a constant spectral distance of K_e. Therefore, under the same conditions our approximations are provably good. Section 2 presents our element-approximation method.

corrected from $n_e/\sqrt{2}$.

The approximability of element matrices by SDD matrices is the second key aspect of our method: the method requires that most (but not necessarily all) of the K_e's be approximable by SDD matrices. This condition, together with the null-space condition on the K_e, are essentially sufficient conditions for our method to be effective. When all the

K_e's have the desired null space and they can all be well approximated by SDD matrices, our solver is provably effective. When a few K_e's cannot be well approximated, the solver is usually effective, but the theoretical analysis is weaker in such cases. More specifically, when some K_e's are inapproximable, the preconditioner may be expensive to factor (but less expensive than K), but the convergence rate does not deteriorate.

After it computes the element-by-element approximations, our solver splits the elements into two sets. One set, denoted $E(t)$, contains all the elements in which the approximation is better than some parameter t, and the other set contains the inapproximable elements. We now scale each L_e so that the sum $L = \sum_{e \in E(t)} \alpha_e L_e$ of scaled approximations is spectrally close to $K_{\leq t} = \sum_{e \in E(t)} K_e$. This step is presented in Section 3.

The matrix L is itself an SDD matrix. Over the last decade, several provably-good combinatorial graph algorithms for constructing preconditioners for SDD matrices have been developed [35, 18, 17, 5, 34, 15, 33, 24]. Some of them, as well as some combinatorial heuristics, have been shown to be effective in practice [12, 21, 29, 30, 16, 28].

Our algorithms use one of these combinatorial algorithms to construct another SDD matrix M that approximates L. The difference between M and L is that M can be factored more quickly into sparse triangular factors than L. We now find a scaling factor γ such that $\gamma M + \sum_{e \notin E(t)} K_e$ is spectrally close to K. The construction of M is explained in Section 4 and the choice of γ is explained in Section 5. Finally, we factor the sum $\gamma M + \sum_{e \notin E(t)} K_e$ and use its factors as a factored preconditioner in a preconditioned symmetric Krylov-subspace solver such as Conjugate Gradients [14, 20], SYMMLQ, or MINRES [26]. For most of the combinatorial algorithms that we can use to construct M, it is possible to show that the preconditioner is spectrally close to K. The spectral bounds give a bound on the number of iterations that the Krylov-subspace algorithm performs.

Putting it all together, we obtain practical and provably-efficient algorithms for solving a large class of scalar elliptic PDEs. By provably-efficient we mean that there is a theoretical bound on the total amount of work performed by the solver. The bounds counts all the aspects of the solution process, including constructing the L_e's and L, constructing M, factoring M, and performing the iterations. The costs associated with the different phases of the solver are described in Section 6.

Experimental results that explore the performance and behavior of our solver are presented in Section 7. These results show that the solver is highly reliable. In particular, we show that on some problems other solvers, including an algebraic-multigrid solver and an

incomplete-Cholesky solver, either fail or are very slow; our solver handles these problems without difficulty.

2. Nearly-Optimal Element-by-Element Approximations

In this section we show how to compute a nearly optimal SDD approximation L_e to a given symmetric positive (semi)definite matrix K_e that is either nonsingular or has a null space consisting only of the constant vector.

2.1. **Defining the Problem.**

Let $\mathbb{S}$ be a linear subspace of $\mathbb{R}^n$ (the results of this section also apply to $\mathbb{C}^n$, but we use $\mathbb{R}^n$ in order to keep the discussion concrete). We denote by $\mathbb{R}^{\mathbb{S}} \subseteq \mathbb{R}^{n \times n}$ the set of symmetric (semi)definite matrices whose null space is exactly $\mathbb{S}$.

Definition 2.1. Given two matrices A and B in $\mathbb{R}^{\mathbb{S}}$, a *finite generalized eigenvalue* λ of (A, B) is a scalar satisfying $Ax = \lambda Bx$ for some $x \notin \mathbb{S}$. The *generalized finite spectrum* $\Lambda(A, B)$ is the set of finite generalized eigenvalues of (A, B), and the *generalized condition number* $\kappa(A, B)$ is

$$\kappa(A, B) = \frac{\max \Lambda(A, B)}{\min \Lambda(A, B)} .$$

(This definition can be generalized to the case of different null spaces, but this is irrelevant for this paper.) We informally refer to $\kappa(A, B)$ as the spectral distance between A and B. We also define the condition number $\kappa(A) = \kappa(A, P_{\perp\mathbb{S}})$ of a single matrix $A \in \mathbb{R}^{\mathbb{S}}$, where $P_{\perp\mathbb{S}}$ is the orthogonal projector onto the subspace orthogonal to $\mathbb{S}$.

We refer to the following optimization problem as the *optimal symmetric semidefinite approximation problem.*

Problem 2.2. Let A be a symmetric positive (semi)definite matrix with null space $\mathbb{S}$ and let $B_1, B_2, \ldots, B_m$ be rank-1 symmetric semidefinite matrices. Find coefficients $d_1, d_2, \ldots, d_m$ that minimize the generalized condition number of A and

$$B_{\text{opt}} = \sum_{j=1}^{m} d_j^2 B_j$$

under the constraint $\text{null}(B_{\text{opt}}) = \text{null}(A)$, or decide that no such coefficients exist. The generalized condition number $\kappa(A, B_{\text{opt}})$ is invariant to scaling of B_{opt}, so we can assume without loss of generality that all the B_j's have unit norm.

2.2. **From Generalized Condition Numbers to Condition Numbers.**

The main tool that we use to find nearly optimal solutions to Problem (2.2) is a reduction of the problem to the well studied problem of scaling the columns of a matrix to minimize its spectral condition number.

A slightly different representation of the problem is useful for characterizing the optimal solution. Let $B_j = Z_j Z_j^T$, where Z_j is a column vector. Let Z be an n-by-m matrix whose columns are the Z_j's. Then

$$\sum_{j=1}^{m} d_j^2 B_j = \sum_{j=1}^{m} d_j^2 Z_j Z_j^T = ZDD^T Z^T$$

where D is the m-by-m diagonal matrix with d_j as its jth diagonal element. This yields an equivalent formulation of the optimal symmetric semidefinite approximation problem.

Problem 2.3. Given a symmetric positive (semi)definite n-by-n matrix A with null space $\mathbb{S}$ and an n-by-m matrix Z, find an m-by-m diagonal matrix D such that $\text{null}(ZDD^T Z^T) = \mathbb{S}$ and that minimizes the generalized condition number $\kappa(A, ZDD^T Z^T)$, or report that no such D exists.

We are interested in cases where $\text{range}(Z) = \text{range}(A)$.

Lemma 2.4. *If A is symmetric and range(Z) = range(A), then Problem (2.3) is feasible.*

Proof. Let D be the n-by-n identity. Then $\text{range}(ZDD^T Z^T) = \text{range}(ZZ^T) = \text{range}(Z) = \text{range}(A)$, so $\text{null}(ZDD^T Z^T) = \text{null}(A) = \mathbb{S}$. □

If $\text{range}(Z) \supsetneq \text{range}(A)$, the problem may or may not be feasible. For example, suppose that the first $m-1$ columns of Z span $\text{range}(A)$ and that the mth column is linearly independent of the first $m-1$. Then clearly

$$Z \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & 0 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & 0 \end{bmatrix}^T Z^T$$

has the same null space as A, so this problem is feasible. On the other hand, if

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \text{and} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

then $\text{null}(ZDD^T Z^T) \neq \text{null}(A)$ for every diagonal D.

The following lemma, which is a generalization of [9, Theorem 4.5], is the key to the characterization of B_{opt}.

Lemma 2.5. *Let $A = UU^T$ and $B = VV^T$ be two matrices in $\mathbb{R}^{\mathbb{S}}$ for some $\mathbb{S}$. We have*

$$\Lambda(A, B) = \Sigma^2 \left(V^+ U\right)$$

and

$$\Lambda(A, B) = \Sigma^{-2} \left(U^+ V\right) .$$

In these expressions, $\Sigma(\cdot)$ is the set of nonnzero singular values of the matrix within the parentheses, Σ^ℓ denotes the same singular values to the ℓth power, and V^+ denotes the Moore-Penrose pseudoinverse of V.

New proof.

Proof. Both U and V have n rows, so U^+ and V^+ have n columns. Therefore, the products V^+U and U^+V exist.

Since A and B have the same null space, U and V must have the same range. Therefore, every column U_i of U is in the range of V, which implies that $V(V^+U_i) = U_i$ [4, Proposition 6.1.7]. Therefore, $VV^+U = U$. We denote $W = V^+U$ and obtain $U = VW$.

Let $\lambda \in \Lambda(A, B)$, let $x \notin \mathbb{S}$ satisfy $Ax = \lambda Bx$, and let $y = V^T x$. Because $x \notin \mathbb{S}$, $Bx = VV^T x \neq 0$, so $y = V^T x \neq 0$. We have

$$\begin{aligned} WW^T y &= V^+UU^T \left(V^+\right)^T V^T x \\ &= V^+U \left(U^T \left(V^+\right)^T V^T\right) x \\ &= V^+U \left(VV^+U\right)^T x \\ &= V^+UU^T x \\ &= V^+Ax \\ &= V^+\lambda Bx \\ &= \lambda V^+VV^T x \\ &= \lambda V^T x \\ &= \lambda y \ . \end{aligned}$$

We have used the indentity $VV^+U = U$ to transition from the third to the fourth lines, and the identity $V^+VV^T = V^T$, which holds for any matrix V, to transition from line seven to eight.

This implies $\lambda \in \Lambda(WW^T) = \Sigma^2(W) = \Sigma^2(V^+U)$.

Now let $\lambda \in \Sigma^2(V^+U) = \Lambda(WW^T)$, let y satisfy $WW^T y = \lambda y$, and let $x = \left(V^T\right)^+ y$. The relation $WW^T y = \lambda y$ implies that y is in the range of W, because $\lambda \neq 0$ (the definition of Σ ensures this). We have $Wz = y$ for some z, so $V^+Uz = y$. Since y is in the range of V^+, it is also in the range of V^T. Therefore, $V^T x = V^T \left(V^T\right)^+ y = y$. We now expand Ax to obtain

$$\begin{aligned} Ax &= UU^T x \\ &= UU^T \left(V^T\right)^+ y \\ &= U \left(V^+U\right)^T y \\ &= UW^T y \ . \end{aligned}$$

We now multiply both sides by V^+ to obtain

$$\begin{aligned} V^+UU^Tx &= V^+UW^Ty \\ &= WW^Ty \\ &= \lambda y \\ &= \lambda V^Tx \ . \end{aligned}$$

We multiply both sides by V and use the equality $VV^+U = U$ to get

$$\begin{aligned} VV^+UU^Tx &= \lambda VV^Tx \\ UU^Tx &= \lambda VV^Tx \\ Ax &= \lambda Bx \ . \end{aligned}$$

so $\lambda \in \Lambda(A, B)$.

The second result $\Lambda(A, B) = \Sigma^{-2}(U^+V)$ follows from replacing the roles of A and B in the analysis above and from the equality $\Lambda(A, B) = \Lambda^{-1}(B, A)$. The reversal yields

$$\Lambda(A, B) = \Lambda^{-1}(B, A) = \left(\Sigma^2\left(U^+V\right)\right)^{-1} = \Sigma^{-2}\left(U^+V\right) \ .$$

□

This lemma shows that Problems (2.2) and (2.3) can be reduced to the problem of scaling the columns of a single matrix to minimize its spectral condition number. Let $A = UU^T$ and let Z satisfy range(Z) = range(A). (If A is symmetric semidefinite but U is not given, we can compute such a U from the Cholesky factorization of A or from its eigendecomposition.) According to the lemma,

$$\Lambda(A, ZDD^TZ^T) = \Sigma^{-2}\left(U^+ZD\right) \ .$$

Therefore, minimizing $\kappa(A, ZDD^TZ^T)$ is equivalent to minimizing the spectral condition number $\kappa(U^+ZD)$ under the constraint range(ZD) = range(Z).

The other set equality in Lemma 2.5 does not appear to be useful for such a reduction. The equality

$$\Lambda(A, ZDD^TZ^T) = \Sigma^2\left((ZD)^+U\right) \ ,$$

but unfortunately, there does not appear to be a way to simplify $(ZD)^+U$ in a way that makes D appear as a row or column scaling. (Note that in general, $(ZD)^+ \neq D^+Z^+$.)

The problem of scaling the columns of a matrix to minimize its spectral condition number has been investigated extensively. Although exact optimization algorithms do not exist, there is a simple approximation algorithm.

2.3. **Computing the Pseudoinverse of a Factor of an Element Matrix.** Before we can find the optimal scaling, we need to compute U^+ from a given element matrix $K_e = UU^T$ and to form U^+Z.

We compute U^+ in one of two ways. If the input to our solver is an element matrix K_e with a known null space, we can compute U^+ from an eigendecomposition of K_e. Let $K_e = Q_e\Lambda_e Q_e^T$ be the *reduced* eigendecomposition of K_e (that is, Q_e is n-by-rank(K_e) and Λ_e is a rank(K_e)-by-rank(K_e) nonsingular diagonal matrix). We have $K_e = Q_e\Lambda_e^{1/2}\left(Q_e\Lambda_e^{1/2}\right)^T$ so we can set $U = Q_e\Lambda_e^{1/2}$, so $U^+ = \Lambda_e^{-1/2}Q_e^T$.

Many finite-elements discretization actually generate the element matrices in a factored form. If that is the case, then some symmetric factor F of $K_e = FF^T$ is given to our solver as input. In that case, we compute a reduced singular-value decomposition SVD of F, $F = Q_e\Sigma_e R_e^T$, where Σ_e is square, diagonal, and invertible, and R_e is square and unitary, both of order rank(F). Since

$$K_e = FF^T = Q_e\Sigma_e R_e^T R_e\Sigma_e^T Q_e^T = Q_e\Sigma_e^2 Q_e^T$$

is an eigendecomposition of K_e, we can set $U = Q_e\Sigma_e$ and we have $K_e = UU^T$. In this case $U^+ = \Sigma_e^{-1}Q_e^T$. This method is more accurate when K_e is ill conditioned.

Note that in both cases we do not need to explicitly form U^+; both methods provide a factorization of U^+ that we can use to apply it to Z.

Once we form U^+Z, our solver searches for a diagonal matrix D that brings the condition number of U^+ZD close to the minimal condition number possible. This problem is better understood when U^+Z is full rank. Fortunately, in our case it always is.

Lemma 2.6. *Let U be a full rank m-by-n matrix, $m \geq n$, and let Z be an m-by-ℓ matrix with range(Z) = range(U). Then U^+Z is full rank.*

Proof. Since range(Z) = range(U), there exists an n-by-l matrix C such that $Z = UC$. By definition, rank(Z) $\leq$ rank(C) $\leq n$. Moreover, since range(Z) = range(U) and U is full rank, we have that n = rank(Z). Therefore, rank(C) = n.

It is sufficient to show that $C = U^+Z$ to conclude the proof of the lemma. Since U is full rank and $m \geq n$, the product U^+U is the n-by-n identity matrix. Therefore, $U^+Z = U^+UC = C$.

□

Without the assumption range(Z) = range(U), the matrix U^+Z can be rank deficient even if both $U^+$and Z are full rank.

Example 2.7. Let

$$U = \begin{bmatrix} 2 & 0 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix}.$$

The columns of U are orthogonal, range$(Z) \neq$ range(U). This gives

$$U^+ = \begin{bmatrix} 1/3 & -1/6 & -1/6 \\ 0 & -1/2 & 1/2 \end{bmatrix}$$

and

$$U^+ Z = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/2 \end{bmatrix} ,$$

which is clearly not full rank.

2.4. **Nearly-Optimal Column Scaling.** Given a matrix U^+Z, we wish to find a diagonal matrix D that minimizes the spectral condition number of U^+ZD, under the assumption range(Z) = range(U), and under the constraint that range(ZD) = range(Z).

To keep the notation simple and consistent with the literature, in this section we use A to denote U^+Z and we use m and n to denote the number of rows and columns in $A = U^+Z$.

The key result that we need is due to van der Sluis [36], who showed that choosing $\tilde{D}$ such that all the columns of $A\tilde{D}$ have unit 2-norm brings $A\tilde{D}$ to within a factor of $\sqrt{n}$ of the optimal scaling. Van der Sluis, extending an earlier result of Bauer for square invertible matrices [2], analyzed the full-rank case.

Given an m-by-n matrix A, $m \geq n$, and a nonsingular diagonal D van der Sluis defined

$$\kappa_{\text{vdS}}(AD) = \frac{\|AD\|_2}{\min_{x \neq 0} \|ADx\|_2 / \|x\|_2} \tag{2}$$

(his original definition is not specific to the 2-norm, but this is irrelevant for us). He, like Bauer, was interested in finding the a diagonal matrix D that minimizes (2). This definition of the problem implicitly assumes that A is full rank, otherwise $\kappa_{\text{vdS}}(AD) = \infty$ for any nonsingular diagonal D. Also, if A is full rank than the minimizing D must give a full-rank AD. If A has more columns than rows, we can use a complementary result by van der Sluis, one that uses row scaling on a matrix with more rows than columns. Van der Sluis result show that if the rows of $\tilde{D}A$ have unit 2-norm, then $\kappa_{\text{vdS}}(\tilde{D}A)$ is within a factor of $\sqrt{m}$ of the minimum possible. This gives us the result that we need, because $\kappa_{\text{vdS}}(AD) = \kappa_{\text{vdS}}(D^T A^T)$. Shapiro later showed that van der Sluis's bound is loose by at most a factor of $\sqrt{2}$ [31].

As we have shown in Lemma 2.6, that matrix $A = U^+Z$ whose columns we need to scale is full rank, so van der Sluis's results apply to it.

We note that further progress has been made in this area for square invertible A's, but it appears that this progress is not applicable to our application when $A = U^+Z$ is rectangular (which is usually the case). Braatz and Morari showed that for a square invertible A, the minimum of $\kappa(AD)$ over all positive diagonal matrices D can be found by solving

a related optimization problem, which is convex [11]. Their paper states that this result also applies to the rectangular case [11, Remark 2.5]; what they mean by that comment is that the related optimization problem minimizes $\|AD\|_2\|D^{-1}A^+\|_2$ [10], whereas we need to minimize $\kappa(AD) = \|AD\|_2\|(AD)^+\|_2$.

2.5. **Nearly-Optimal Symmetric Diagonally-Dominant Approximations.** We now turn our attention to the matrices that arise as element matrices in finite-elements discretizations of (1). Such a matrix K_e has null space that is spanned by the constant vector and by unit vectors e_j for every zero column j of K_e. The part of the null space that is spanned by the unit vectors is irrelevant, so we assume that we are dealing with a matrix A whose null space is spanned by constant vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

We wish to approximate a symmetric semidefinite matrix A with this null space (or possibly a nonsingular matrix) by a symmetric diagonally-dominant matrix B,

$$B_{ii} \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |B_{ij}| \ .$$

To define a matrix Z such that the expression ZDD^TZ^T can generate any symmetric diagonally-dominant matrix, we define the following vectors.

Definition 2.8. Let $1 \leq i, j \leq n$, $i \neq j$. A length-n *positive edge vector*, denoted $\langle i, -j\rangle$, is the vector

$$\langle i,-j\rangle = \begin{matrix} \\ i \\ \\ j \\ \\ \end{matrix}\!\!\begin{bmatrix} \vdots \\ +1 \\ \vdots \\ -1 \\ \vdots \end{bmatrix}, \quad \langle i,-j\rangle_k = \begin{cases} +1 & k = i \\ -1 & k = j \\ 0 & \text{otherwise.} \end{cases}$$

A *negative edge vector* $\langle i, j\rangle$ is the vector

$$\langle i,j\rangle = \begin{matrix} \\ i \\ \\ j \\ \\ \end{matrix}\!\!\begin{bmatrix} \vdots \\ +1 \\ \vdots \\ +1 \\ \vdots \end{bmatrix}, \quad \langle i,j\rangle_k = \begin{cases} +1 & k = i \\ +1 & k = j \\ 0 & \text{otherwise.} \end{cases}$$

A *vertex vector* $\langle i\rangle$ is the unit vector

$$\langle i\rangle_k = \begin{cases} +1 & k = i \\ 0 & \text{otherwise.} \end{cases}$$

A symmetric diagonally dominant matrix can always be expressed as a sum of outer products of scaled edge and vertex vectors. Therefore, we can conservatively define Z to be the matrix whose columns are all the positive edge vectors, all the negative edge vectors, and all the vertex vectors.

If A is singular and its null space is the constant vector, we can do better. Chen and Toledo provided a combinatorial characterization of the null space of SDD matrices [13].

Lemma 2.9. *[Chen and Toledo] Let A be a symmetric diagonally-dominant matrix whose null space is the constant vector. Then A is a sum of outer products of scaled positive edge vectors. Furthermore, the null space of a symmetric diagonally-dominant matrix with a positive off-diagonal element (corresponding to an outer product of a scaled negative edge vector) cannot be span $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.*

Therefore, if A is singular with this null space, we only need to include in the column set Z the set of positive edge vectors. If A is nonsingular, we also include in Z negative edge vectors and vertex vectors.

We can also create even sparser Z's; they will not allow us to express every SDD B as $B = ZDD^TZ^T$, but they will have the same null space as A. To define these sparser Z's, we need to view the edge vector $\langle i, -j\rangle$ as an edge that connects vertex i to vertex j in a graph whose vertices are the integers 1 through n. The null space of ZZ^T is the constant vector if an only if the columns of Z, viewed as edges of a graph, represent a connected graph. Therefore, we can build an approximation $B = ZDD^TZ^T$ by selecting an arbitrary connected graph on the vertices $\{1, \ldots, n\}$. By [13, Lemma 4.2], if A is nonsingular, we can include in Z the positive edge vectors of a connected graph plus one arbitrary vertex vector.

If A is well conditioned (apart perhaps from one zero eigenvalue), we can build a good approximation $B = ZDD^TZ^T$ even without the column-scaling technique of Lemma 2.5. In particular, this avoids the computation of the pseudo-inverse of a factor U of $A = UU^T$. Clearly, if A is nonsingular and well conditioned, then we can use I as an approximation: the generalized condition number $\kappa(A, I)$ is $\kappa(A)$. If A has rank $n-1$ and the constant vector is its null vector, than

$$B_{C(n_e)} = \frac{1}{n_e}\begin{bmatrix} n_e - 1 & -1 & -1 & \cdots & -1 \\ -1 & n_e - 1 & -1 & \cdots & -1 \\ -1 & -1 & n_e - 1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & \cdots & n_e - 1 \end{bmatrix}$$

yields

$$\kappa(A, B_{C(n_e)}) = \sigma_1(A)/\sigma_{n-1}(A)\ , \tag{3}$$

which we assumed is low (we denote the singular values by σ_j in non increasing order). The matrix $B_{C(n_e)}$ is the Laplacian of the complete graph, and it is clearly SDD. The identity (3) follows from the fact that $B_{C(n_e)}$is an orthogonal projection onto range(A). The following lemma summarizes this discussion.

Lemma 2.10. *Let A be a symmetric positive (semi)definite matrix. If A is nonsingular, or if the null space of A is span $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$, then there is an* SDD *matrix B such that $\kappa(A, B) \leq \kappa(A)$.*

This result may seem trivial (it is), but it is nonetheless important. The global stiffness matrix $K = \sum_e K_e$ is often ill conditioned, but the individual element matrices K_e are usually well conditioned. Since we build the approximation $L = \sum_e L_e$ element by element, this lemma is often applicable: When K_e is well conditioned, we can set L_e to be an extension of $B_{C(n_e)}$ into an n-by-n matrix. The rows and columns of the scaled $B_{C(n_e)}$ are mapped to rows and columns $\mathcal{N}_e$ of L_e and the rest of L_e is zero.

For well-conditioned A's, we can also trade the approximation quality for a sparser approximation than $B_{C(n_e)}$. The matrix

$$B_{S(n_e)} = \frac{1}{n_e} \begin{bmatrix} n_e - 1 & -1 & -1 & \cdots & -1 \\ -1 & 1 & 0 & \cdots & 0 \\ -1 & & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ -1 & & 0 & \cdots & -1 \end{bmatrix}$$

gives

(4)

$$\kappa(A, B_{S(n_e)}) \leq \kappa(A, B_{C(n_e)})\kappa(B_{C(n_e)}, B_{S(n_e)}) = \kappa(A)\kappa(B_{S(n_e)}) = \frac{n_e \sigma_1(A)}{\sigma_{n-1}(A)} .$$

For small n_e, this may be a reasonable tradeoff. The bound (4) follows from the observation that the eigenvalues of $B_{S(n_e)}$ are exactly 0, 1, and n_e.

When A is ill conditioned, there may or may not be an SDD matrix B that approximates it well. The following two examples demonstrate both cases.

Example 2.11. Let

$$A = \frac{1}{2\epsilon} \begin{bmatrix} 1 + \epsilon^2 & -\epsilon^2 & -1 \\ -\epsilon^2 & \epsilon^2 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

for some small $\epsilon > 0$. This matrix has rank 2 and null vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$, and it its condition number is proportional to $1/\epsilon^2$. Since A diagonally-dominant, there is clearly an SDD matrix B (namely, A itself) such that $\kappa(A, B) = 1$. This matrix is the element matrix for a linear

triangular element with nodes at $(0,0)$, $(0,\epsilon)$, and $(1,0)$ and a constant $\theta = 1$. The ill conditioning is caused by the high aspect ratio of the triangle, but this ill conditioned matrix is still diagonally dominant. Small perturbations of the triangle will yield element matrices that are not diagonally dominant but are close to a diagonally dominant one.

Example 2.12. Our example of a matrix that cannot be approximated well by an SDD matrix is the element matrix for an isosceles triangle with two tiny angles and one that is almost π, with nodes at $(0,0)$, $(1,0)$, and $(1/2,\epsilon)$ for some small $\epsilon > 0$. The element matrix is

$$A = \frac{1}{2\epsilon} \begin{bmatrix} \frac{1}{4}+\epsilon^2 & \frac{1}{4}-\epsilon^2 & -\frac{1}{2} \\ \frac{1}{4}-\epsilon^2 & \frac{1}{4}+\epsilon^2 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$$

This matrix has rank 2 and null vector $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$. We now show that for any SDD matrix B with the same null space, $\kappa(A,B) \geq \epsilon^{-2}/4$. Let $v = \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}^T$ and $u = \begin{bmatrix} 1 & 1 & -2 \end{bmatrix}^T$; both are orthogonal to $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$. We have $v^T A v = 2\epsilon$ and $u^T A u = 4.5\epsilon^{-1}$. Therefore,

$$\begin{aligned} \kappa(A,B) &= \max_{x \perp \mathrm{null}(A)} \frac{x^T A x}{x^T B x} \times \max_{x \perp \mathrm{null}(A)} \frac{x^T B x}{x^T A x} \\ &\geq \frac{u^T A u}{u^T B u} \times \frac{v^T B v}{v^T A v} \\ &= \frac{4.5}{2\epsilon^2} \times \frac{v^T B v}{u^T B u} \, . \end{aligned}$$

We denote the entries of B by

$$B = \begin{bmatrix} b_{12}+b_{13} & -b_{12} & -b_{13} \\ -b_{12} & b_{12}+b_{23} & -b_{23} \\ -b_{13} & -b_{23} & b_{13}+b_{23} \end{bmatrix}$$

where the b_{ij}'s are non-negative. Furthermore, at least two of the b_{ij}'s must be positive, otherwise B will have rank 1 or 0, not rank 2. In particular, $b_{13} + b_{23} > 0$. This gives

$$\frac{v^T B v}{u^T B u} = \frac{4b_{12}+b_{13}+b_{23}}{9b_{13}+9b_{23}} = \frac{4b_{12}}{9b_{13}+9b_{23}} + \frac{1}{9} \geq \frac{1}{9} \, .$$

Therefore, $\kappa(A,B) > \epsilon^{-2}/4$, which can be arbitrarily large.

2.6. **A Heuristic for Symmetric Diagonally-Dominant Approximations.** In Section 2.5 we have shown how to find a nearly-optimal SDD approximation B to a symmetric positive (semi)definite matrix A whose null space is spanned by $\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$. In this section we show a simple heuristic. We have found experimentally that it often works well. On the other hand, we can show that in some cases it produces

approximations that are arbitrarily far from the optimal one. Thus, this section has two related goals: to describe a simple heuristic that often works well, but to point out that it cannot always replace the method of Section 2.5.

Definition 2.13. Let A be a symmetric positive (semi)definite matrix. We define A_+ to be the SDD matrix defined by

$$(A_+)_{ij} = \begin{cases} a_{ij} & i \neq j \text{ and } a_{ij} < 0 \\ 0 & i \neq j \text{ and } a_{ij} \geq 0 \\ \sum_{k \neq j} -(A_+)_{ik} & i = j \ . \end{cases}$$

Clearly, A_+ is SDD.

If the null space of A is spanned by $\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$, the matrix $-(A - A_+)$ is also SDD. It turns out that in many cases, $\kappa(A, A_+)$ is small, making A_+ a good approximation for A. In particular, it is possible to show that $\kappa(A, A_+) \leq n\kappa(A)/4$ (we omit the proof), so if A is well conditioned and small, then A_+ is always a fairly good approximation. The matrix A_+ is also sparser than A, which is also helpful.

But when A is ill conditioned, A_+can be an arbitrarily bad approximation even though A is approximable by some other SDD matrix.

Example 2.14. Let $0 < \epsilon \ll 1$, and let $M \geq \frac{4}{\epsilon}$,

$$A = \begin{bmatrix} 1+M & -1 & 0 & -M \\ -1 & 1+M & -M & 0 \\ 0 & -M & M & 0 \\ -M & 0 & 0 & M \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\epsilon & -1+\epsilon \\ 0 & 0 & -1+\epsilon & 1-\epsilon \end{bmatrix} .$$

This matrix is symmetric semidefinite with rank 3 and null vector $\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$. We show that for small ϵ, A is ill conditioned, with condition number larger than $8\epsilon^{-2}$. Let

$$q_1 = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} , \quad q_2 = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} , \quad q_3 = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} , \text{ and } \quad q_4 = \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

be an orthonormal basis for $\mathbb{R}^4$. We have

$$\begin{aligned} q_1^T A q_1 &= 0 \\ q_2^T A q_2 &= 2M \\ q_3^T A q_3 &= 2M + \epsilon \\ q_4^T A q_4 &= \epsilon \ . \end{aligned}$$

Therefore, $\kappa(A) \geq 2M/\epsilon \geq 8\epsilon^{-2}$. We show that the matrix A_+ is a poor approximation of A.

$$\begin{aligned} q_1^T A_+ q_1 &= 0 \\ q_2^T A_+ q_2 &= 2M \\ q_3^T A_+ q_3 &= 2M+1 \\ q_4^T A_+ q_4 &= 1 \, . \end{aligned}$$

Therefore,

$$\kappa(A, A_+) > \left(1 - \frac{1-\epsilon}{2M+1}\right) \epsilon^{-1} \approx \epsilon^{-1} \, .$$

On the other hand, the SDD matrix

$$B = \begin{bmatrix} \epsilon + M & -\epsilon & 0 & -M \\ -\epsilon & \epsilon + M & -M & 0 \\ 0 & -M & \epsilon + M & -\epsilon \\ -M & 0 & -\epsilon & \epsilon + M \end{bmatrix}$$

is a good approximation of A, with $\kappa(A, B) < 9$. This bound follows from a simple path-embedding arguments [3], which shows that $3A - B$ and $3B - A$ are positive semidefinite. The quantitative parts of these arguments rest on the inequalities

$$\frac{1}{2M} + \frac{1}{2M} + \frac{1}{3-\epsilon} \leq \frac{1}{3-4\epsilon}$$

and

$$\frac{1}{2M} + \frac{1}{2M} + \frac{1}{1+2\epsilon} < \frac{1}{1-3\epsilon} \, ,$$

which hold for small ϵ.

3. Scaling and Assembling Element-by-Element Approximations

Given a set of approximations $\{L_e\}$ to a set of element matrices $\{K_e\}$, our solver scales the L_e's so that their assembly $L = \sum_e \alpha_e L_e$ is a good approximation of $K = \sum_e K_e$. We can scale them in one of two equivalent ways. The next lemma shows that under these scalings, if even L_e is a good approximation of K_e, then L is a good approximation of K. Both scaling rules require the computation of one extreme eigenvalue for each pair (K_e, L_e).

Lemma 3.1. *Let $\{K_e\}_{e \in E}$ and $\{L_e\}_{e \in E}$ be sets of symmetric positive (semi)definite matrices with null(K_e) = null(L_e). Let $\alpha_e = \min \Lambda(K_e, L_e)$*

and let $\beta_e = \max \Lambda(K_e, L_e)$. *Then*

$$\begin{aligned} \kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\leq \max \kappa\left(K_e, L_e\right) \\ \kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \beta_e L_e\right) &\leq \max \kappa\left(K_e, L_e\right) \end{aligned}$$

Proof. Scaling L_e by α_e transforms the smallest generalized eigenvalue of $\Lambda(K_e, \alpha_e L_e)$ to 1, since

$$\Lambda(K_e, \alpha_e L_e) = \frac{1}{\alpha_e}\Lambda(K_e, L_e) \ .$$

Scaling L_e clearly does not change the generalized condition number, so $\max \Lambda(K_e, \alpha_e L_e) = \kappa(K_e, L_e)$.

By the splitting lemma [3],

$$\begin{aligned} \min \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\geq \min\left\{\min \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \min\left\{1\right\}_{e\in E} \\ &= 1 \ . \end{aligned}$$

Also by the splitting lemma,

$$\begin{aligned} \max \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\leq \max\left\{\max \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \max\left\{\kappa\left(K_e, L_e\right)\right\}_{e\in E} \ . \end{aligned}$$

Combining these two inequalities gives the result. The proof for scaling by β_e is the same. □

If we use inexact estimates $\tilde{\alpha_e}$ for the minimum of $\Lambda(K_e, L_e)$, the bound becomes

$$\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right) \leq \frac{\max_e\left\{(\alpha_e/\tilde{\alpha_e})\,\kappa\left(K_e, L_e\right)\right\}}{\min_e\left(\alpha_e/\tilde{\alpha_e}\right)} \ ,$$

and similarly for estimates of β_e. This shows that $\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right)$ depends on how much the estimates vary. In particular, if the relative estimation errors are close to 1, scaling by the estimates is almost as good as scaling by the exact eigenvalues.

4. Combinatorial Sparsification of the Assembled SDD Approximation

Once we obtain an SDD approximation $L = \sum_e \alpha_e L_e$ of K, we can use a combinatorial graph algorithm to construct an easier-to-factor SDD approximation M of L. Because M is spectrally close to L and L is spectrally close to K, M is also spectrally close to K. By applying [9,

Proposition 3.6] to both ends of the generalized spectra, we obtain the following lemma.

Lemma 4.1. *Let K, L, and M be symmetric (semi)definite matrices with the same dimensions and the same null space. Then*

$$\kappa(K, M) \leq \kappa(K, L)\kappa(L, M) \ .$$

There are several algorithms that can build M from L. All of them view an exactly (but not strictly) SDD matrix L as a weighted undirected graph G_L, to which they build an approximation graph G_M. The approximation G_M is then interpreted as an SDD matrix M. If L is strictly diagonal dominant, the approximation starts from an exactly SDD matrix $L - D$, where D is diagonal with nonnegative elements. From $L - D$ the algorithm builds an approximation $\tilde{M}$; If $\tilde{M}$ is a good approximation to $L - D$, then $\tilde{M} + D$ is a good approximation to L.

Lemma 4.2. *Let A and B be symmetric positive (semi)definite matrices with the same null space $\mathbb{S}$, and let C be a positive symmetric (semi)definite with a null space that is a subspace, possibly empty, of $\mathbb{S}$. Then*

$$\Lambda(A + C, B + C) \subseteq [\min \Lambda(A, B) \cup \{1\}, \max \Lambda(A, B) \cup \{1\}] \ .$$

Proof. The result is a simple application of the splitting lemma [3], since

$$\begin{aligned} \max \Lambda(A + C, B + C) &\leq \max \{\max \Lambda(A, B), \max \Lambda(C, C)\} \\ &= \max \{\max \Lambda(A, B), 1\} \\ &= \max \Lambda(A, B) \cup \{1\} \ , \end{aligned}$$

and similarly for the smallest generalized eigenvalue. □

This lemma is helpful, since most of the algorithms that construct approximations of SDD matrices provide theoretical bounds on $\Lambda(L - D, \tilde{M})$ that have 1 as either an upper bound or a lower bound. When this is the case, adding D to $L - D$ and to $\tilde{M}$ preserves the theoretical condition-number bound.

The earliest algorithm for this subproblem is Vaiyda's algorithm [35, 3, 12]. This algorithm finds a maximum spanning tree in G_M and augments it with suitable edges. The quantity of extra edges that are added to the tree is a parameter in this algorithm. When few or no edges are added, $\kappa(L, M)$ is high (but bounded by $nm/2$, where m is the number of off-diagonal elements in L), but L is cheap to factor (can be factored in $O(n)$ operations when no edges are added to the tree). When many edges are added, $\kappa(L, M)$ shrinks but the cost of factoring L rises. When G_M is planar or nearly planar then the algorithm is very effective, both theoretically and experimentally. For more general classes of graphs, even with a bounded degree, the algorithm is less effective.

A heuristic for adding edges to the maximum spanning tree was proposed by Franngioni and Gentile [16]. They designed their algorithm for SDD linear systems that arise in the interior-point solution of network-flow problems. There are no theoretical convergence bounds for this algorithm.

Several algorithms are based on so-called *low-stretch spanning trees*. Boman and Hendrickson realized that a low-stretch spanning tree G_M is of G_L leads to a better convergence-rate bound than maximum spanning trees [8]. Elkin et al. showed simpler and lower-stretch constructions for low-stretch trees [15]. Spielman and Teng proposed algorithms that create denser graphs G_M (which are still based on low-stretch trees) [34, 33]. The algorithms of Spielman and Teng lead to nearly-linear work bound for solving $Lx = b$.

Another class of algorithms construct rooted balanced trees with more than n vertices; the leaves of these trees are the vertices of G_L (the rows/columns of L). These trees were first invented by Gremban and Miller [18, 17], and a more general construction was recently proposed by Maggs et al. [23]. Because these trees are balanced, the iterative solver parallelizes essentially perfectly even though it solves two triangular linear systems per iteration.

5. Dealing with Inapproximable Elements

When some of the element matrices cannot be approximated well by an SDD matrix, we split the global stiffness matrix K into $K = K_{\le t} + K_{>t}$, where $K_{\le t} = \sum_{e \in E(t)} K_e$ is a sum of the element matrices for which we found an SDD approximation L_e that satisfies $\kappa(K_e, L_e) \le t$ for some threshold $t > 0$, and $K_{>t} = \sum_{e \notin E(t)} K_e$ is a sum of element matrices for which our approximation L_e gives $\kappa(K_e, L_e) > t$.

We then scale the approximations in $E(t)$ and assemble them to form $L_{\le t} = \sum_{e \in E(t)} \alpha_e K_e$. We now apply one of the combinatorial graph algorithms discussed in Section 4 to construct an approximation $M_{\le t}$ to $L_{\le t}$. Once we have $M_{\le t}$, we add it to $K_{>t}$ to obtain a preconditioner $M_1 = M_{\le t} + K_{>t}$.

This construction gives a bound on $\kappa(K, M_1)$, but it is a heuristic in the sense that M_1 may be expensive to factor. The analysis of $\kappa(K, M_1)$ is essentially the same as the analysis of strictly-dominant matrices in Section 4: by Lemma 4.2, a theoretical bound $\Lambda(K_{\le t}, M_{\le t}) \subseteq [\alpha, \beta]$ implies $\Lambda(K, M_1) \subseteq [\min\{\alpha, 1\}, \max\{\beta, 1\}]$.

The scaling technique of Lemma 3.1 ensures that either $\alpha, \beta \le 1$ or $1 \le \alpha, \beta$. But the interval $[\alpha, \beta]$ may be quite far from 1. If the interval is far from 1, $\kappa(K, M_1)$ can be large. To avoid this danger, we scale $M_{\le t}$ before adding it to $K_{>t}$; that is, we use a preconditioner $M_\gamma = \gamma M_{\le t} + K_{>t}$. We choose γ as follows. We find some vector v that is orthogonal to $\text{null}(M_{\le t})$ and compute its generalized Raleigh

quotient

$$\gamma = \frac{v^T K_{\leq t} v}{v^T M_{\leq t} v} .$$

The null space of $M_{\leq t}$ is determined by the connected components of its graph, so it is easy to quickly find such a v (we use a random v in this subspace). This definition ensures that $\gamma \in [\alpha, \beta]$. Since $\Lambda(K_{\leq t}, \gamma M_{\leq t}) \subseteq [\alpha/\gamma, \beta/\gamma]$, we have and $1 \in [\alpha/\gamma, \beta/\gamma]$.

Lemma 5.1. *Under this definition of M_γ, $\kappa(K, M_\gamma) \leq \beta/\alpha$, where the interval $[\alpha, \beta]$ bounds $\Lambda(K_{\leq t}, M_{\leq t})$. In particular, if we take α and β to be the extremal generalized eigenvalues of $(K_{\leq t}, M_{\leq t})$, we obtain*

$$\kappa(K, M_\gamma) \leq \kappa(K_{\leq t}, M_{\leq t}) .$$

We expect that this overall heuristic will be effective when $E \setminus E(t)$ contains only few elements. If only a few elements cannot be approximated, then $K_{>t}$ is very sparse, so the sparsity pattern of $M_\gamma = \gamma M_{\leq t} + K_{>t}$ is similar to that of $M_{\leq t}$. Since $M_{\leq t}$ was constructed so as to ensure that its sparsity pattern allow it to be factored without much fill, we can expect the same to hold for M_γ. If $E \setminus E(t)$ contains many elements, there is little reason to hope that the triangular factor of M_γ will be particularly sparse.

6. Asymptotic Complexity Issues

In this section we explain the asymptotic complexity of the different parts of our solver. We do not give a single asymptotic expression that bounds the work that the solver performs, but comment on the cost and asymptotics of each phase of the solver. The cost of some of the phases is hard to fully analyze, especially when $E(t) \subsetneq E$. The next section presents experimental results that complement the discussion here.

The first action of the solver is to approximate each element matrix K_e by an SDD matrix $\alpha_e L_e$. For a given element type, this phase scales linearly with the number of elements and it parallelizes perfectly. The per-element cost of this phase depends on the approximation method and on the number n_e of degrees of freedom per element. Asymptotically, all the approximation methods require n_e^3 operations per element, but the uniform-clique is the fastest method. This phase also gives us $\kappa(K_e, \alpha_e L_e)$ which we use to decide which elements belong to $E(t)$ and which do not.

The next phase of the solver assembles the scaled SDD approximations in $E(t)$. The cost of this step is bounded by the cost to assemble $K = \sum_e K_e$, which most finite-elements solvers (including ours), perform. The assemblies of K and L performs $O(\sum_e n_e^2)$ operations: fewer than the first phase, but harder to parallelize.

The cost and the asymptotic complexity of the sparsification of L depends on the algorithm that is used. For Vaidya's sparsification algorithm, which our code uses, the amount of work is $O(n \log n + \sum_e n_e^2)$. For the algorithm of Spielman and Teng [34, 33], the work is $O(m^{1.31})$ where $m = \sum_e n_e^2$.

Next, the algorithm assembles the element matrices K_e that are not in $E(t)$ into $M_{\leq t}$. The cost of this phase is also dominated by the cost of assembling K.

The cost of computing the Cholesky factorization of M is hard to characterize theoretically, because the cost depends on the nonzero pattern of M in a complex way. The nonzero pattern of M depends on how many and which elements are not in $E(t)$, and on how much we sparsified $L_{\leq t}$. The number of operations in a phase of the solver is not the only determinant of running time, but also the computational rate. The Cholesky factorization of M usually achieves high computational rates.

The cost of the iterative solver depends on the number of iterations and on the per-iteration cost. The amount of work per iteration is proportional to the number of nonzeros in K plus the number of nonzeros in the Cholesky factor of M. The sparsification algorithms of Vaidya and Spielman and Teng control the number of iterations, and if $E(t) = E$ than they also control the density of the Cholesky factor.

7. Experimental Results

This section presents experimental results that explore the effectiveness of our solver.

7.1. **Setup.** Our solver currently runs under MATLAB [25], but it is implemented almost entirely in C. The C code is called from MATLAB using MATLAB's CMEX interface. The element-by-element approximations are computed by C code that calls LAPACK [1]. The assembly of the element-by-element approximations (and possibly the inapproximable elements) is also done in C. The construction of Vaidya's preconditioners for SDD matrices is done by C code [12]. The Cholesky factorization of the preconditioner is computed by MATLAB's sparse `chol` function, which in MATLAB 7.2 calls CHOLMOD 1.0 by Tim Davis. We always order matrices using METIS version 4.0 [22] prior to factoring them. The iterative Krylov-space solver that we use is a preconditioned Conjugate Gradients written in C and based on MATLAB's `pcg`.; within this iterative solver, both matrix-vector multiplications and solution of triangular linear systems are performed by C code.

In most experiments we compare our solver to an algebraic multigrid solver, BoomerAMG [19]. We use the version of BoomerAMG that is packaged as part of HYPRE 1.2. We compiled it using GCC version 3.3.5, with options `-O` (this option is HYPRE's default compilation option).

TABLE 1. Notation for the test problems.

The Domain Ω	
C	A 3-dimensional cube
B	A 3-dimensional box with aspect ratio 1-by-1-by-10000
CH	A 1-by-1-by-1 cube with a 1-by-0.1-by-0.79 hole in the middle
SC	A 10-by-10-by-10 cube containing a spherical shell of inner radius 3 and thickness 0.1.
The Mesh (the parameter indicates the number n *of mesh points)*	
G	3-dimensional, generated by TETGEN
D	3-dimensional, generated by DISTMESH
The Conductivity $\theta(x)$	
U	uniform and isotropic, $\theta = I$ everywhere
J	jump between subdomains but uniform and isotropic within subdomain (e.g., $\theta = 10^4 I$ in the spherical shell of domain SC and $\Theta = I$ elsewhere); the parameter indicates the magnitude of the jump
A	anisotropic within a subdomain (e.g. the spherical shell in SC) and $\theta = I$ elsewhere; θ is always 1 in the x and y directions and the parameter indicates the conductivity in the z direction.
The element type	
L	Linear tetrahedral element, 4-by-4 element matrix
Q	Quadratic tetrahedral element, 10-by-10 element matrix

In some experiments we compare our solver to solvers that are based on incomplete Cholesky preconditioners. To compute these preconditioners, we use MATLAB's built-in `cholinc` routine. Here too, the matrices are preordered using METIS.

Since many of our test problems are ill conditioned, we iterate until the relative residual is at most 10^{-14}, close to $\epsilon_{\text{machine}}$, in order to achieve acceptable accuracy.

We use two mesh generators to partition the three-dimensional problem domains into finite elements. We usually use TETGEN version 1.4. [32]. In a few experiments we DISTMESH [27], which can generate both two- and three-dimensional meshes.

Running times were measured on a 1.6 GHz AMD Opteron 242 computer with 8 GB of main memory, running Linux 2.6. This computer has 2 processors, but our solver only uses one. We used a 64-bit version of MATLAB 7.2. This version of MATLAB uses the vendor's BLAS, a library called ACML. The measured running times are wall-clock times that were measured using the `ftime` Linux system call.

7.2. **Test Problems.** We evaluated our solver on several three-dimensional problems. We used both trilinear and quadratic tetrahedral elements. Table 1 summarizes the problems that we used in the experiments. The boundary conditions are always pure Neumann $\partial u/\partial n = 0$, and we removed the singularity by fixing the solution at a fixed unknown (algebraically, we remove the row and column of K corresponding to that unknown). We generate the right-hand side b of the linear system $Kx = b$ by generating a random solution vector x and multiplying it by K to form b.

In all the experiments reported below, our solver produced acceptable forward errors. The computed solution $\hat{x}$ always satisfies

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq 10^{-4} .$$

7.3. **Choosing the Parameter t.** We begin with simple problems that are designed to help us choose t, the approximation threshold. The behavior of our solver depends on two parameters, t and the aggressiveness of the combinatorial sparsification algorithm. These parameters interact in complex ways, because both influence the sparsity of the Cholesky factor of M and the number of iterations in the Krylov-space solver. It is hard to visualize and understand the performance of a solver in a two-dimensional (or higher) parameter space. Therefore, we begin with experiments whose goal is to establish a reasonable value for t, a value that we use in all of the subsequent experiments.

Figure 1 shows the results of these experiment, which were carried out on two meshes, one generated by DISTMESH and the other by TETGEN. The K_e are trilinear, and their approximations L_e are nearly-optimal cliques. The graphs on the left show the distributions of $\kappa(K_e)$ and $\kappa(K_e, L_e)$. With DISTMESH, we see that the elements belong to two main groups, a large group of elements with generalized condition numbers smaller than about 100, and a small set of so-called slivers with much higher condition numbers, ranging from 200 to 10^8. It appears that for the non-slivers, $\kappa(K_e, L_e)$ is smaller than $\kappa(K_e)$ by roughly a constant factor. For the slivers, $\kappa(K_e, L_e)$ is close to $\kappa(K_e)$. With TETGEN, there are no highly ill-conditioned elements, and the distributions of $\kappa(K_e)$ and $\kappa(K_e, L_e)$ are smoother.

The graphs on the right show the number of iterations that the Conjugate Gradients algorithm performs for several values of t and various levels of sparsification. In all the graphs in the paper whose horizontal axis is fill in the Cholesky factor, the horizontal axis ranges from 0 to the number of nonzeros in the Cholesky factor of K. When t is small, $K_{>t}$ is relatively dense, so the sparsification algorithm cannot be very effective. Even when we instruct Vaidya's algorithm to sparsify $L_{\leq t}$ as much as possible, the Cholesky factor of M remains fairly similar to the Cholesky factor of K. On the other hand, a small t leads to faster

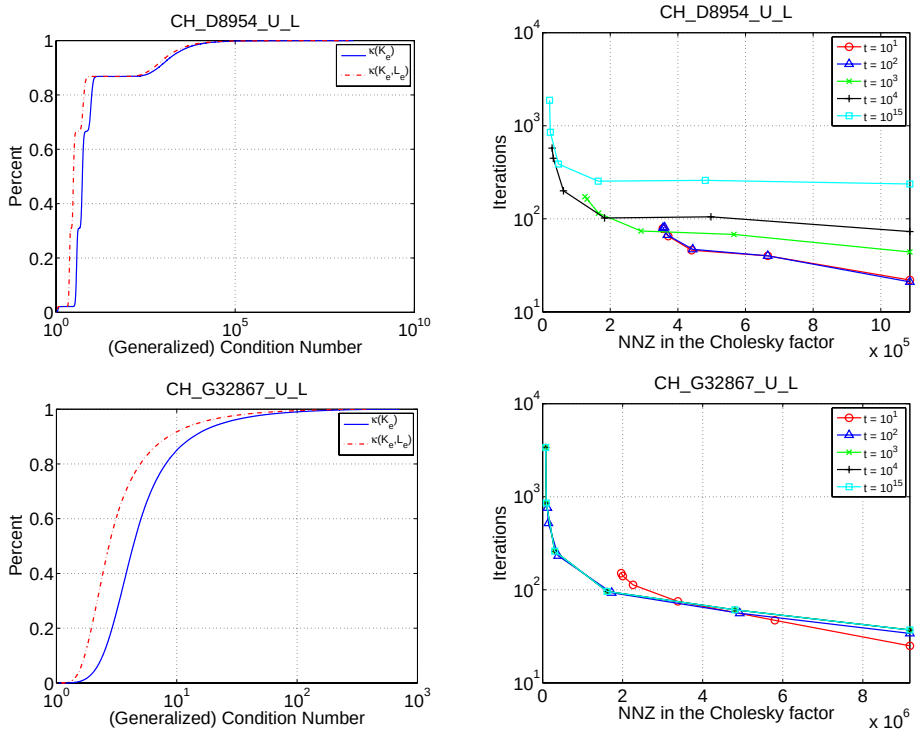


FIGURE 1. The distribution of element condition numbers and approximation qualities (left graphs) and the number of iterations for several values of t and several sparsification levels (right graphs). The top row shows results on a mesh generated by DISTMESH, and the bottom row on a mesh generated by TETGEN.

convergence. With a large t we can construct M's with very sparse factors, but convergence is very slow. A value of t near 1000 gives a good balance between high iteration counts caused by using L_e's with fairly high $\kappa(K_e, L_e)$ and the inability to construct a sparse preconditioner caused by a dense $K_{>t}$. We use $t = 1000$ in the remaining experiments.

7.4. **Baseline Tests.** The next set of experiments shows the performance of our solver relative to other solvers on the same problems and the same meshes, for a few relatively easy problems. The graphs in Figure 2 compare the running time of our solver to that of an incomplete Cholesky preconditioner, BoomerAMG, and a state-of-the-art direct solver, CHOLMOD. In these graphs the vertical axis represents wall-clock time for all the phases of the solution and the horizontal axis represents the number of nonzeros in the triangular factors of the preconditioner. The rightmost (largest) horizontal coordinate in the graphs always corresponds to the number of nonzeros in a complete sparse Cholesky factor of the coefficient matrix. When the complete

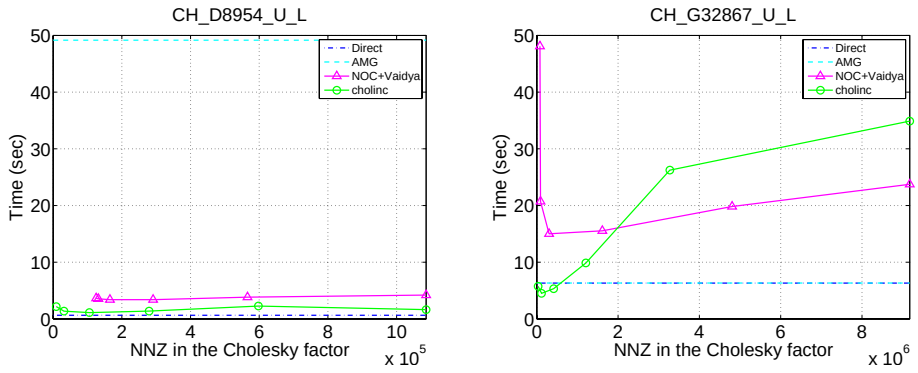


FIGURE 2. Running times for our solver, for incomplete Cholesky, for BoomerAMG, and for a direct solver on simple 3-dimensional problems. The graph on the left uses a mesh generated by DISTMESH, and the one on the right a mesh generated by TETGEN. See the first paragraph of Section 7.4 for a complete explanation of the graphs.

factorization runs out of space, we still use this scaling of the horizontal axis, and we estimate the running time of the complete factorization based on the assumptions that it runs at 10^9 floating-point operations per second. The direct solver and BoomerAMG only give us one data point for each problem; their running times are represented in the graphs by horizontal lines. We ran each preconditioned solver with several values of the parameter that controls the sparsity of the factor (drop tolerance in incomplete Cholesky and the sparsification parameter in Vaidya's preconditioner). Therefore, for each preconditioned solver we have several data points that represented by markers connected by lines. Missing markers and broken lines indicate failures to converge within a reasonable amount of time. Most of the remaining graphs in this section share the same design.

The graphs in Figure 2 compare the running time of the solvers on easy problems with a relatively simple domain and uniform coefficients. The mesh produced by TETGEN leads to a linear system that is easy for all the iterative solvers. With a good drop-tolerance parameter, incomplete Cholesky is the fastest, with little fill. Our solver is slower than all the rest, even with the best sparsity parameter. The mesh produced by DISTMESH causes problems to BoomerAMG, but incomplete Cholesky is faster than our solver.

Although the performance of incomplete Cholesky appears to be good in the experiments reported in Figure 2, it sometimes performs poorly even on fairly simple problems. Figure 3 shows that on a high-aspect-ratio 3-dimensional structure with uniform coefficients, incomplete Cholesky performs poorly: the sparser the preconditioner, the

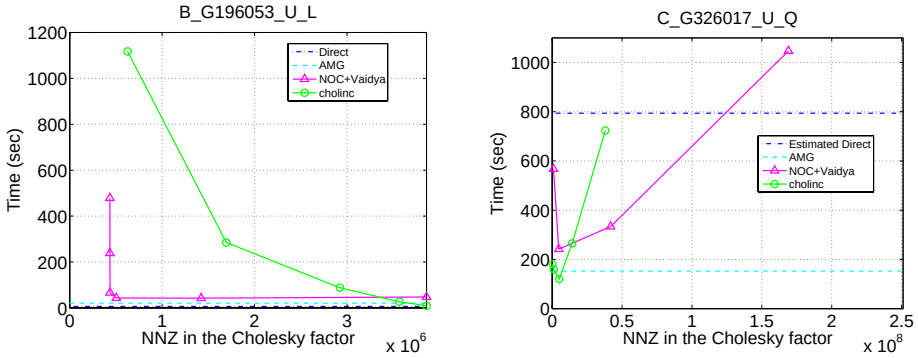
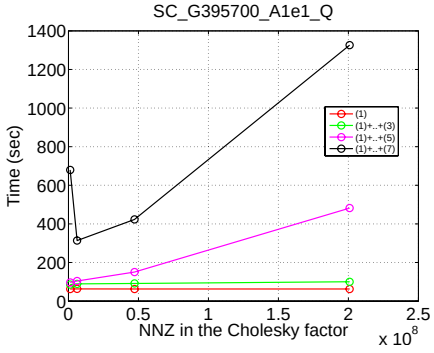


FIGURE 3. Experimental results on two additional problems with uniform coefficients; these problems are much larger than those analyzed in Figure 2.



Notation for phases of the solver:
(1) Approximate K_e by L_e
(2) Assembly $L_{\le t} = \sum_{E(t)} L_e$
(3) Sparsify $L_{\le t}$ to obtain $M_{\le t}$
(4) Assembly of $M = M_{\le t} + K_{>t}$
(5) Order, permute, and factor M
(6) Assembly of $K = \sum_E K_e$
(7) Permute K and iterate

FIGURE 4. A breakdown of the running time of our solver, on a particular problem. The graph shows the time consumed by the different phases of the solver. Assembly phases are not separately shown because their running time is negligible.

slower the solver. Our solver, on the other hand, performs reasonably well even when its factor is much sparser than the complete factor. On the high-aspect-ratio problem, as well as on any problem of small to moderate size, the direct solver performs well. But as the problem size grows the direct solver becomes slow and tends to run out of memory. The rightmost graph in Figure 3 shows a typical example.

Figure 4 shows a breakdown of the running time of our solver for one particular problem. The data shows that as the preconditioner gets sparse, the time to factor the preconditioner decreases. The running time of the iterative part of the solver also initially decreases, because the preconditioner gets sparser. This more than offsets the growth in the number of iterations. But when the preconditioner becomes very sparse, it becomes less effective, and the number of iterations rises quickly.

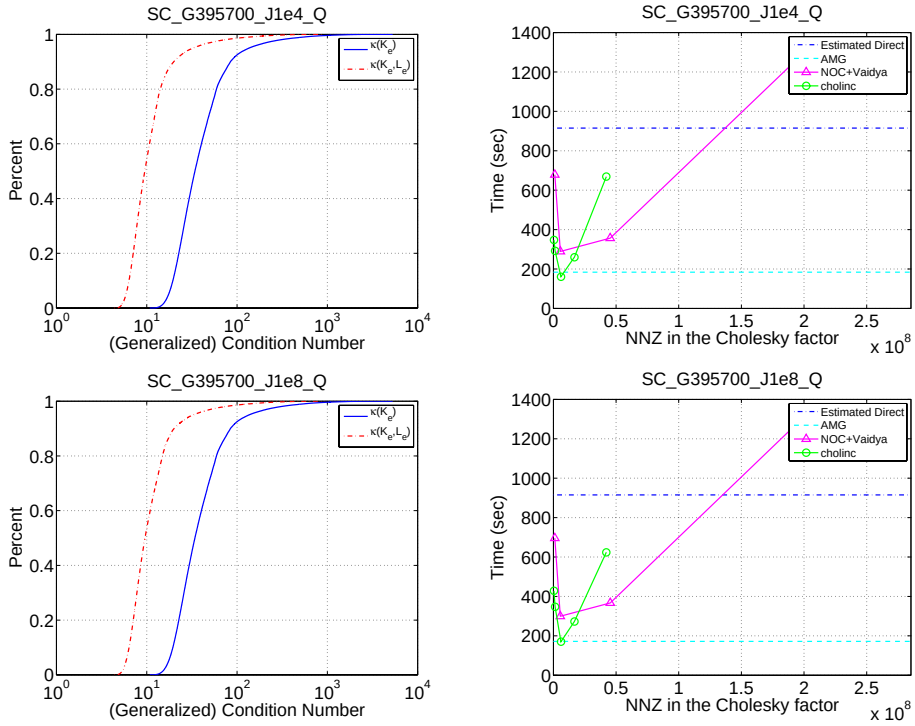


FIGURE 5. Running times and element conditioning for problems with jumping coefficients.

7.5. **Well-Conditioned Elements and Jumping Coefficients.** The next set of experiments explores problems with a large jump in the conductivity θ. We instructed the mesh generators to align the jump with element boundaries, so within each element, there is no jump. This leads to a large $\kappa(K)$, but the conditioning of individual element matrices is determined by their geometry, not by θ. The results, shown in Figure 5, show that the jump in θ does not influence any of the four solvers in a significant way.

7.6. **Ill-Conditioned Elements: Anisotropy.** Some of the experiments shown in Section 7.3 included ill-conditioned elements. The ill-conditioning of those elements resulted from their geometrical shape. Other mesh generators may be able to avoid such element shapes. Indeed, TETGEN did not produce such elements, only DISTMESH did. But in problems that contain anisotropic materials in complex geometries, ill-conditioned elements are hard to avoid.

Figure 6 compares the performance of our solver with that of other solvers on a problem in which the conductivity θ is anisotropic in one part of the domain. The results clearly show that anisotropy leads to ill-conditioned element matrices. As the anisotropy increases, Boomer-AMG becomes slower and incomplete Cholesky becomes less reliable. The anisotropy does not have a significant influence on our solver.

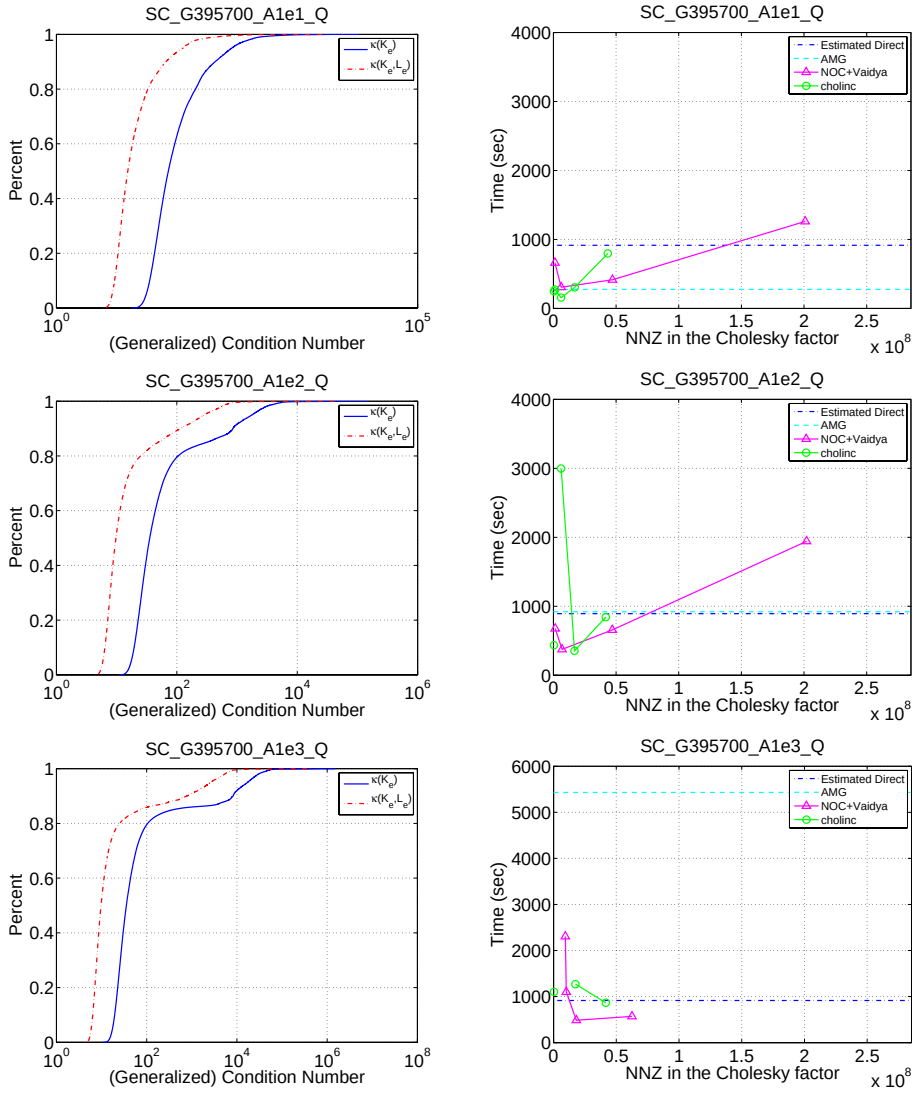


FIGURE 6. The behavior of our solver and other solvers on a solid 3-dimensional problem that contains a thin spherical shell with anisotropic material. The mesh was generated by TETGEN. In the top row, the anisotropy is 10, in the middle row 10^2, and on the bottom row it is 10^3.

In experiments not reported here, our solver behaved well even with anisotropy of 10^8. The incomplete-factorization solver becomes not only slow, but also erratic, as the graphs in Figure 7 show. The convergence of our preconditioner is always steady, monotonically decreasing, and the convergence rate is monotonic in the density of the preconditioner. The convergence of incomplete Cholesky is erratic, not always monotonic, sometimes very slow. Furthermore, sometimes one

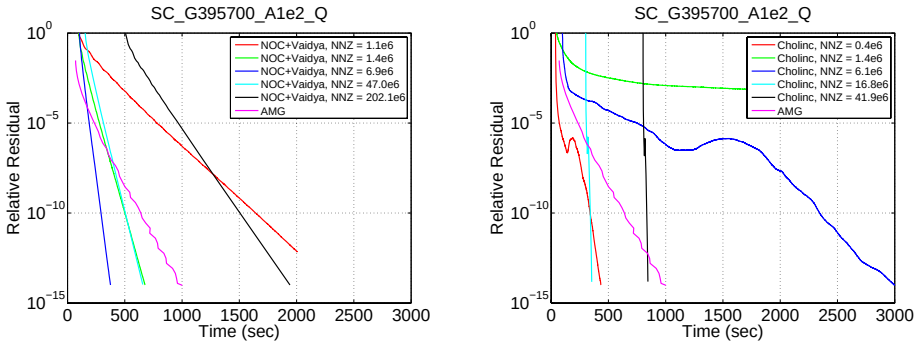


FIGURE 7. The relative norm of the residual as a function of the running time (and implicitly, of the iteration count) on one anisotropic problem, for our solver (left) and for incomplete Cholesky (right). The horizontal coordinate in which individual plots start indicates the time to construct and factor the preconditioner.

incomplete factor leads to much faster convergence than a much denser incomplete factor.

These results show that the ability of our solver to detect highly-ill-conditioned elements (it actually detects inapproximable elements, but the sets largely coincide) and to treat them separately allows it to solve problems that other iterative solvers cannot. When there are few such elements, as there are here because the anisotropic shell is thin, these elements do not significantly increase the density of the factor of M. In problems in which most of the elements are inapproximable by SDD matrices, M would be similar to K and the characteristics of our solver would be similar to the characteristics of a direct solver.

This is perhaps the most important insight about our solver. As problems get harder (in the sense that more elements become inapproximable), its behavior becomes closer to that of a direct solver. As problems get harder we lose the ability to effectively sparsify the preconditioner prior to to factoring it. But unlike other solvers, our solver does not exhibit slow or failed convergence on these difficult problems.

7.7. **Comparisons of Different Element-by-Element Approximations.** We now explore additional heuristics for approximating K_e. The approximation methods that we compare are:

Nearly Optimal Clique (NOC): $L_e = ZDD^TZ$, where the columns of Z is the full set of edge vectors and D scales the columns of U^+Z to unit 2-norm. This method gives the strongest theoretical bound on $\kappa(K_e, L_e)$: it is at most n_e times larger than the best possible for an SDD approximation of K_e. Here and in the next four methods, we set the scaling factor α_e to be $\max \Lambda(K_e, L_e)$.

Nearly Optimal Star (NOS): $L_e = ZDD^TZ$, where the columns of Z are edge vectors that form a star, $\langle 1, -2\rangle, \langle 1, -3\rangle, \ldots, \langle 1, -n_e\rangle$ and D scales the columns of U^+Z to unit 2-norm. Sparser than the first but usually a worse approximation.

Uniform Clique (UC): L_e is the extension of the n_e-by-n_e matrix $B_{C(n_e)}$ to an n-by-n matrix. Computing L_e is cheap, but the approximation is only guaranteed to be good when A is very well conditioned. The low cost of this method stems from the facts that (1) $B_{C(n_e)}$ is a fixed matrix, and (2) $\alpha_n = \max \Lambda(K_e)$, so we do not need to estimate an extreme generalized eigenvalue, only a single-matrix eigenvalue.

Uniform Star (US): L_e is the extension of the n_e-by-n_e matrix $B_{S(n_e)}$ to an n-by-n matrix. Sparser than the uniform clique, but more expensive, since we compute an extreme generalized eigenvalue to set α_e.

Positive Part (PP): $L_e = (K_e)_+$, defined in Section 2.6.

Boman et al. (BHV): $\alpha_e L_e$ is the Boman-Hendrickson-Vavasis approximation of K_e [7]. In their method, L_e is a uniform star, and the scaling factor α_e is computed from quantities associated with the finite-element discretization that produces K_e.

The results are shown in Figure 8. When element matrices are fairly well conditioned (bottom graphs), different approximation methods exhibit similar qualitative behaviors. But when there are ill-conditioned elements, naive approximation methods perform poorly. The results also show that the nearly-optimal approximation (which we used in all the other experiments) performs well relative to other approximation methods, but is usually not the best.

7.8. **Running Times for Growing Problem Sizes.** The results in Figure 9 present the growth in running time of our solver as the problem size grows. For very dense and very sparse preconditioners, the growth is highly nonlinear. This is consistent with the theory of sparse direct solvers on one side and with the theory of Vaidya's preconditioners on the other. For intermediate levels of fill, running times grow more slowly, but they still seem to grow superlinearly.

8. Conclusions and Open Problems

We have presented the first practical support preconditioner for linear systems arising from partial differential equations. Finding such preconditioners has been an important research problem for more than a decade. Vaidya's initial discovery of combinatorial preconditioners for symmetric diagonally-dominant linear systems in the early 1990's [35] motivated additional research on the topic. Researchers quickly discovered, however, that the fact that combinatorial graph preconditioners are applicable only to SDD linear systems is severely limiting. Although

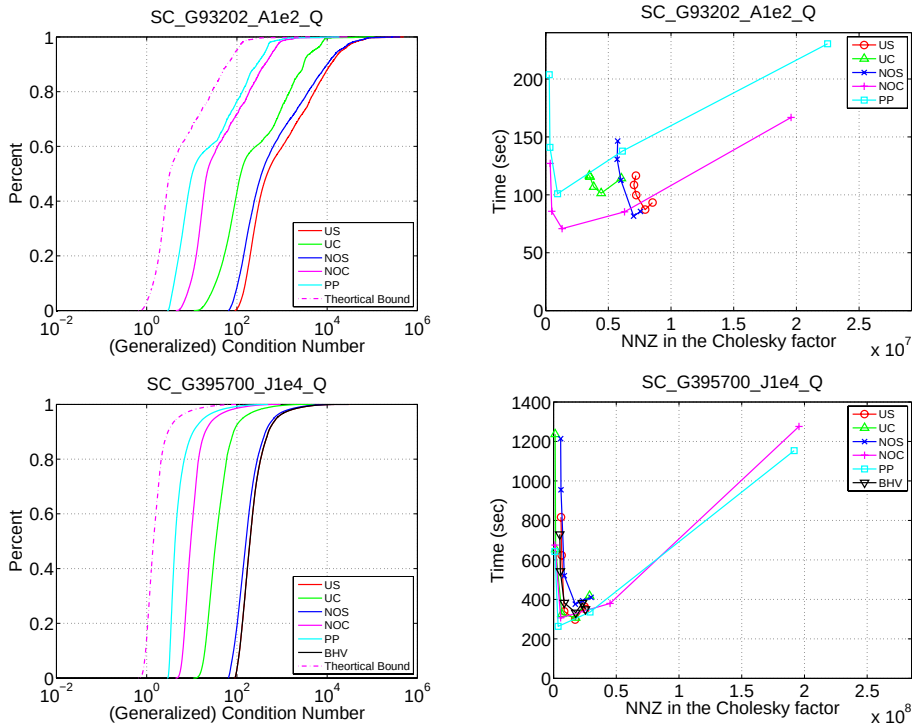


FIGURE 8. Different element-approximation methods. The method of Boman et al. is only applicable to well-conditioned elements, so it is not included in the graphs in the top row.

some finite-differences discretizations of PDE's do lead to SDD linear systems, most finite-elements and many finite-differences discretizations lead to linear systems that are not diagonally-dominant. Generalizing combinatorial graph preconditioners to a wider class of linear systems has been a subject of intense research. Until 2004, the attempts to expand the applicability of combinatorial graph preconditioners led to several discoveries, but these discoveries did not lead to useful preconditioners. Examples of such discoveries include the combinatorial characterization of the so-called H-matrices and their null space [6, 13]. Finally, the discovery in 2004 that element matrices in finite-elements discretizations of scalar elliptic PDE's are often approximable by SDD matrices [7] led to the practical solver that this paper presents.

In the development of our solver we have used many insights gained during the decade-long quest to find practical combinatorial preconditioners. For example, our characterization of generalized eigenvalues in Lemma 2.5 is a generalization of an earlier extremal result [9, Theorem 4.5]; we have repeatedly used the splitting lemma [17, 3] (e.g., in Lemma 4.2 and Section 5); we have used path arguments and support arguments [17, 3] to analyze the approximation in Example 2.14; and

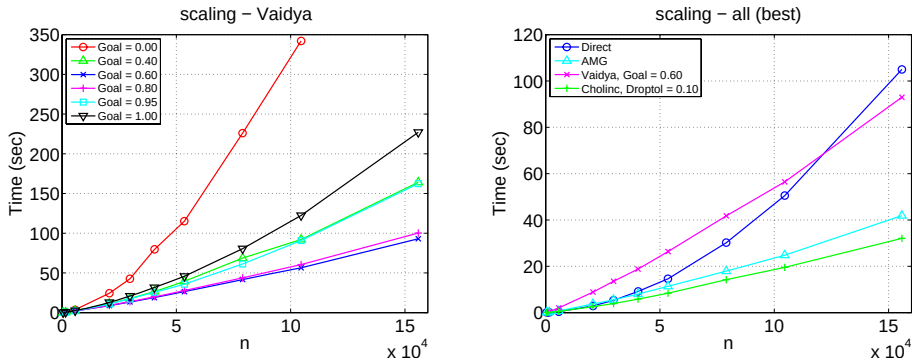


FIGURE 9. Solution times as a function of mesh size, on the same physical problem (a cube with uniform coefficients, discretized using trilinear elements). The graph on the left compares the running times of our solver with different levels of fill, and the graph on the left compares our solver (with the best-case fill level) with Boomer-AMG and the direct solver. The fill in our solver is controlled by a parameter called goal in these graphs. A goal of 1 does not sparsify the approximation $M_{\leq t}$, and a goal of 0 sparsifies it as much as possible, resulting in a tree or forest graph structure for $L_{\leq t}$.

we have used a result about the null space SDD matrices to show that we do not need negative edge vectors as columns of Z in Section 2.5.

Two main technical contributions allowed us to develop this solver. The first is the splitting $K = K_{\leq t} + K_{>t}$ and the realization that we can often construct effective preconditioners by approximating and sparsifying $K_{\leq t}$ but leaving $K_{>t}$ as is. The time and space usage of such preconditioners depend, of course, on the nonzero structure of $K_{>t}$, but they are always reliable. This splitting of K clearly shows that combinatorial preconditioners are fundamentally different from other classes of solvers, such as incomplete-Cholesky, sparse approximate inverses, multigrid and domain decomposition preconditioners. There is no obvious way to use our splitting $K = K_{\leq t} + K_{>t}$ with these other classes of solvers. The new nearly-optimal methods to algebraically approximate element matrices by SDD matrices constitute the second major contribution of this paper. These new methods generate better approximations and are more general than the approximation technique of Boman, Hendrickson, and Vavasis [7].

This paper raises several interesting questions and challenges for further research. We mention three. One challenge is to extend the optimal-scaling method of Braatz and Morari [11] to rank-deficient and rectangular matrices. It is not even clear whether van der Sluis's nearly-optimal scaling for rectangular matrices [36] is also nearly-optimal for

rank-deficient matrices. Another interesting question is to find a reliable and cheap-to-compute estimate of the spectral distance between a given symmetric positive (semi)definite matrix A and the closest SDD matrix to A. We have shown that $\kappa(A)$ is an upper bound on that distance, but we have also shown that this bound can be arbitrarily loose. The third and probably most important challenge is to find better ways to exploit the splitting $K = K_{\leq t} + K_{>t}$. There may be several ways to exploit it. For example, it is probably possible to build better preconditioners by sparsifying $L_{\leq t}$ with the objective to reduce fill in the Cholesky factor of $M_{\leq t} + K_{>t}$; the algorithm that we used for the sparsification phase ignores $K_{>t}$ and only tries to reduce fill in the factor of $M_{\leq t}$.

Acknowledgement. This research was supported by an IBM Faculty Partnership Award, by grant 848/04 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by grant 2002261 from the United-States-Israel Binational Science Foundation.

Sivan Toledo's work on this problem started a decade ago, in 1996, when he was working with John Gilbert under DARPA contract DABT63-95-C-0087, "Portable parallel preconditioning". The contributions and support of John and DARPA are gratefully acknowledged.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, PA, 3nd edition, 1999. Also available online from `http://www.netlib.org`.

[2] F. L. Bauer. Optimally scaled matrices. *Numerische Mathematik*, 5:73–87, 1963.

[3] Marshall Bern, John R. Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27:930–951, 2006.

[4] Denis S. Bernstein. *Matrix Mathematics: Theory, Facts, and Formulas with Applications to Linear Systems Theory*. Princeton University Press, 2005.

[5] Erik G. Boman, Doron Chen, Bruce Hendrickson, and Sivan Toledo. Maximum-weight-basis preconditioners. *Numerical Linear Algebra with Applications*, 11:695–721, 2004.

[6] Erik G. Boman, Doron Chen, Ojas Parekh, and Sivan Toledo. On the factor-width and symmetric H-matrices. *Numerical Linear Algebra with Applications*, 2005. To appear.

[7] Erik G. Boman, Bruce Henderickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. Submitted for publication, 2004.

[8] Erik G. Boman and Bruce Hendrickson. On spanning tree preconditioners. Unpublished manuscript, Sandia National Laboratories, 2001.

[9] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3):694–717, 2004.

[10] Richard D. Braatz. Response to Chen and Toledo on "minimizing the Euclidean condition number". Private communication, September 2005.

[11] Richard D. Braatz and Manfred Morari. Minimizing the Euclidean condition number. *SIAM Journal on Control and Optimization*, 32:1763–1768, 1994.

[12] Doron Chen and Sivan Toledo. Vaidya's preconditioners: Implementation and experimental study. *Electronic Transactions on Numerical Analysis*, 16:30–49, 2003.

[13] Doron Chen and Sivan Toledo. Combinatorial characterization of the null spaces of symmetric H-matrices. *Linear Algebra and its Applications*, 392:71–90, 2004.

[14] Paul Concus, Gene H. Golub, and Dianne P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In James R. Bunch and Donald J. Rose, editors, *Sparse Matrix Computations*, pages 309–332. Academic Press, New York, 1976.

[15] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the 37th annual ACM symposium on Theory of computing (STOC)*, pages 494–503, Baltimore, MD, 2005. ACM Press.

[16] A. Frangioni and C. Gentile. New preconditioners for KKT systems of network flow problems. *SIAM Journal on Optimization*, 14:894–913, 2004.

[17] Keith D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, October 1996. Available as Technical Report CMU-CS-96-123.

[18] Keith D. Gremban, Gary L. Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 65–69. IEEE Computer Society, 1995. A longer version is available as Technical Report CMU-CS-94-205, Carnegie-Mellon University.

[19] Van Emden Henson and Ulrike Meier Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[20] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[21] Joaquim J. Júdice, João Patricio, Luis F. Portugal, Mauricio G. C. Resende, and Geraldo Veiga. A study of preconditioners for network interior point methods. *Computational Optimization and Applications*, 24:5–35, 2003.

[22] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998.

[23] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Solving symmetric diagonally-dominant systems by preconditioning. Unpublished manuscript available online at `http://www.cs.cmu.edu/~bmm`, 2002.

[24] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 416–427, Cambridge, Massachusetts, October 2003. IEEE Computer Society.

[25] The MathWorks. Matlab version 7.2. software package, January 2006.

[26] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[27] Per-Olof Persson and Gilbert Strang. A simple mesh generator in MATLAB. *SIAM Review*, 46:329–345, 2004.

[28] L. Portugal, F. Bastos, J. Júdice, J. Paixao, and T. Terlaky. An investigation of interior-point algorithms for the linear transportation problem. *SIAM Journal on Scientific Computing*, 17:1202–1223, 1996.

[29] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Júdice. A truncated primal-infeasible dual-feasible interior point network flow method. *Networks*, 35:91–108, 2000.

[30] M. Resense and G. Veiga. An efficient implementation of the network interior-point method. In D. Johnson and C. McGeoch, editors, *Network Flows and Matching: the First DIMACS Implementation Challenge*, volume 12 of *DIMACS Series in Discrete Mathematics and Computer Science*. AMS.

[31] A. Shapiro. Upper bounds for nearly optimal diagonal scaling of matrices. *Linear and Multilinear Algebra*, 29:145–147, 1991.

[32] Hang Si. *TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator: Users's Manual for Version 1.4*, January 2006. available online from `http://tetgen.berlios.de`.

[33] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. Manuscript; available at `http://www.arxiv.org/abs/cs.DS/0310051`, March 2003.

[34] Daniel A. Spielman and Shang-Hua Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $0(m^{1.31})$. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 416–427, October 2003.

[35] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript. A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computations, Minneapolis, October 1991.

[36] A. van der Sluis. Condition numbers and equilibration of matrices. *Numerische Mathematik*, 14:14–23, 1969.