Electronic Transactions on Numerical Analysis.
Volume 16, pp. 30-49, 2003.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

VAIDYA'S PRECONDITIONERS: IMPLEMENTATION AND EXPERIMENTAL STUDY*

DORON CHEN† AND SIVAN TOLEDO‡

Abstract. We describe the implementation and performance of a novel class of preconditioners. These preconditioners were proposed and theoretically analyzed by Pravin Vaidya in 1991, but no report on their implementation or performance in practice has ever been published. We show experimentally that these preconditioners have some remarkable properties. We show that within the class of diagonally-dominant symmetric matrices, the cost and convergence of these preconditioners depends almost only on the nonzero structure of the matrix, but not on its numerical values. In particular, this property leads to robust convergence behavior on difficult 3-dimensional problems that cause stagnation in incomplete-Cholesky preconditioners (more specifically, in drop-tolerance incomplete Cholesky without diagonal modification, with diagonal modification, and with relaxed diagonal modification). On such problems, we have observed cases in which a Vaidya-preconditioned solver is more than 6 times faster than an incomplete-Cholesky-preconditioned solver, when we allow similar amounts of fill in the factors of both preconditioners. We also show that Vaidya's preconditioners perform and scale similarly or better than drop-tolerance relaxed-modified incomplete Cholesky preconditioners on a wide range of 2-dimensional problems. In particular, on anisotropic 2D problems, Vaidya's preconditioners deliver robust convergence independently of the direction of anisotropy and the ordering of the unknowns. However, on many 3D problems in which incomplete-Cholesky-preconditioned solvers converge without stagnating, Vaidya-preconditioned solvers are much slower. We also show how the insights gained from this study can be used to design faster and more robust solvers for some difficult problems.

Key words. linear-equation solvers, iterative solvers, preconditioning, support preconditioning, support theory, maximum-spanning trees, experimental study.

AMS subject classifications. 65-05, 65F10, 65F35, 65F50, 65N22, 05C05, 05C50, 05C85.

1. Introduction.

A decade ago Pravin Vaidya proposed an intriguing family of preconditioners for symmetric diagonally-dominant (SDD) matrices [27]. He presented his ideas in a scientific meeting but never published a paper on the topic. Neither he nor others ever described an implementation or an experimental study of these preconditioners.[1]

We have implemented Vaidya's preconditioners. We experimentally compare the effectiveness of Vaidya's preconditioners to that of incomplete-factorization preconditioners, including no-fill and drop-tolerance preconditioners, both modified and unmodified. The objective of the comparison is *not* to study the behavior of incomplete Cholesky but to provide a qualitative and quantitative context for our experiments with Vaidya's preconditioners. Our main findings are that Vaidya's preconditioners:

- converge at an almost constant rate on a variety of problems, a behavior very different from the convergence behavior of incomplete Cholesky preconditioners.

*Received September 2, 2001. Accepted for publication January 3, 2003. Recommended by Tony Chan.

†School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

‡School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: `stoledo@tau.ac.il`. Home page: `http://www.tau.ac.il/~stoledo`. This research was supported by Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (grant number 572/00 and grant number 9060/99), by an IBM Faculty Partnership Award, and by the University Research Fund of Tel-Aviv University.

[1]We note that Vaidya founded a company called Computational Applications and System Integration Inc. (`www.casicorp.com`). Pravin Vaidya is the president of the company and a former PhD student of his, Anil Joshi, whose thesis [16] contains some analysis of Vaidya's preconditioners, is the company's vice president. According to its marketing materials, the company is developing and licensing proprietary iterative linear solvers. Two papers describe the performance of the code [23, 24] but without any details on the algorithms that the code uses. Therefore, it is possible that Vaidya's algorithms have been implemented in this proprietary software, but nothing has been published in the scientific literature. Also, since these two papers [23, 24] provide no algorithmic details, it is impossible to determine whether they actually describe the performance of Vaidya's preconditioners or of some other algorithm or variant.

ETNA
Kent State University
etna@mcs.kent.edu

- are sensitive only to the nonzero structure of the coefficient matrix and not to the values of its entries (within the class of SDD matrices), again a different behavior than that of incomplete Cholesky.
- deliver similar or better performance than that of incomplete Cholesky on a wide variety of two-dimensional problems.
- deliver poorer performance than incomplete Cholesky on some three-dimensional problems but dramatically better performance on other three dimensional problems.

Vaidya proposed constructing a preconditioner M for an SDD matrix A by dropping off-diagonal nonzeros from A and factoring M (completely). John Gilbert coined the term *complete factorization of incomplete matrices* to describe such preconditioners, as opposed to conventional incomplete factorizations of the complete matrix. (The same term also describes some existing preconditioners, such as block Jacobi preconditioners.) Vaidya proposed a sophisticated dropping algorithm that balances the amount of fill in M with the condition number of the preconditioned matrix $M^{-1/2}AM^{-1/2}$. Perhaps the most remarkable theoretical aspect of Vaidya's preconditioners is that for many classes of SDD matrices, the condition number of the preconditioned matrix depends only on the size of A and is independent of the condition number $\kappa(A)$ of A.

In this paper we focus on applying Vaidya's preconditioners to a subclass of SDD matrices, the class of SDD matrices with nonpositive offdiagonal elements. A nonsingular SDD matrix with nonpositive offdiagonal elements is a Stieltjes matrix, so from now on we refer to this class of matrices as diagonally-dominant Stieltjes matrices, which we denote by SPDDD (symmetric positive-definite diagonally-dominant). Many of the SDD matrices that arise in applications belong to the class of SDD matrices. When restricted to SPDDD matrices, Vaidya's preconditioners are fairly simple to describe, analyze and construct. Vaidya's preconditioners for general SDD matrices are analyzed in [9].

We now describe how Vaidya's preconditioners for an SPDDD matrix are constructed. Let A be an SPDDD matrix and let G_A be the underlying graph of A. The underlying graph $G_A = (V_A, E_A)$ of an n-by-n symmetric matrix A is a weighted undirected graph whose vertex set is $V_A = \{1, 2, \ldots, n\}$ and whose edge set is $E_A = \{(i,j) : i \neq j \text{ and } A_{i,j} \neq 0\}$; The weight of an edge (i,j) is $-A_{i,j}$. Given a parameter t, the method works by constructing a maximum spanning tree T of G_A and splitting T into roughly t connected components of roughly the same size. The method then adds the heaviest edge in G_A between every pair of subtrees if there is an edge between them in G_A. If there are ties and one of the candidate edges is in T, then this edge is added.

A large value for the parameter t results in a small condition number, and hence convergence in a small number of iterations, at the expense of significant fill in the factors of M. (In particular, $t = n$, where n is the dimension of A, leads to $M = A$.) Fill in the factors slows down the factorization of M and slows down each iteration. A small t on the other hand, yields a higher condition number but sparser factors.

Vaidya stated (without a proof; for a proof, see [6]) that for any n-by-n SPDDD matrix with m nonzeros, $t = 1$ yields a condition number $\kappa = O(mn)$. The construction of such a preconditioner costs only $O(m + n \log n)$ work and its factorization costs only $O(m)$ work. For general sparse SPDDD matrices with a constant bound on the number of nonzeros per row, the theoretically optimal value of t is around $n^{1/4}$, which yields a total solution time (construction, factorization, and iterations) of $O(n^{1.75})$. For SPDDD matrices whose graphs are planar, the total solution time is only $O(n^{1.2})$ when t is chosen appropriately.

This paper consists of two main parts. The first part, presented in Section 2, describes Vaidya's preconditioners in more detail and presents an algorithm for splitting the maximum spanning tree, a detail which was missing from Vaidya's unpublished manuscript.

The second part of the paper, presented in Section 3, presents the results of an experimental study of Vaidya's preconditioners. Since this is the first such study, our goal in designing the experiments was to answer two questions:

- What are the particular strengths and weaknesses of these preconditioners?
- Are Vaidya's preconditioners worthy of further study and investigation?

These questions are important for several reasons. The first question is important since the answer can direct users of Vaidya's preconditioners to problems where it performs well and since the answer can direct researchers to the weaknesses of the method, which perhaps can be addressed using enhancements and modifications. We show below, as an example, how a simple observation about a weakness of Vaidya's preconditioners allows us to improve their performance and use them more appropriately. The second question is important because if these preconditioners do have useful strengths, then it is likely that additional research would be necessary to characterize their behavior more precisely, and that additional research could lead to further enhancements. Our understanding of other families of preconditioners, such as incomplete Cholesky or sparse approximate inverses, is typically the result of tens or hundreds of research papers, not one or two. The second question is also important due to the novelty of Vaidya's preconditioners (they are not simply variants of previously-proposed preconditioners) and from the remarkable theoretical results that Vaidya has been able to prove, such as the $O(n^{1.2})$ total work bound for 2-dimensional SPDDD problems, no matter how large $\kappa(A)$. Not every novel idea that is theoretically appealing translates into a practical numerical method, but such ideas do deserve to be tested in practice, which is what we do in this paper.

Our study attempts to answer these questions by experimentally comparing the performance of Vaidya's preconditioners to that of drop-tolerance incomplete-Cholesky preconditioners (IC), in the context of a conjugate gradients iterative solver. (More specifically, we compare Vaidya's preconditioners to drop-tolerance IC preconditioners without modification, with modification, and with relaxed modification.) The comparisons to IC preconditioners are used only to provide a familiar qualitative and quantitative context to the performance metrics of Vaidya's preconditioners and to highlight their strengths and weaknesses. **Our experiments are not meant to reveal any particular strength or weakness in IC preconditioners, and the results should not be understood in that way**. Furthermore, although our experiments may appear to reveal weaknesses in IC preconditioners, some of these weaknesses can be addressed by other variants of incomplete factorizations, such as incomplete factorizations by level of fill (ILU(k); as opposed to drop tolerance), robust incomplete factorizations [1], dynamic incomplete factorizations [22], and so on.

Our interpretation of the experimental results, which are presented in Section 3, are as follows. Vaidya's preconditioners are insensitive to the numerics of the problem. Their performance does not vary much when we change the boundary conditions of the underlying PDEs, when we change the direction of anisotropy in anisotropic problems, or when we introduce large discontinuities in the coefficients of the underlying PDEs. Vaidya's preconditioners are sensitive to the nonzero structure of the coefficient matrices, and in particular, they perform better on large 2D problems than on large 3D problems. Both of these observations are predicted by the theoretical analysis of the preconditioners, which seems to hold in practice in spite of the finite-precision arithmetic. In contrast, IC preconditioners are highly sensitive to the numerics of the problem. In particular, on very difficult problems IC preconditioners stagnate for hundreds or thousands of iterations before converging, even when large amounts of fill are allowed. We have not observed a similar stagnation phenomenon with Vaidya's preconditioners. A numerical computation of the spectra of some Vaidya- and IC-preconditioned operators helps explain these phenomena. Our conclusions from the ex-

ETNA
Kent State University
etna@mcs.kent.edu

periments are summarized in more detail in the concluding section of the paper, Section 4.

2. Vaidya's Preconditioners.

2.1. Theory and Construction. In this section we describe the preconditioners that Vaidya proposed and the algorithms that we used to construct them. Vaidya's method constructs a preconditioner M whose underlying graph G_M is a subgraph of G_A. The graph G_M of the preconditioner has the same set of vertices as G_A and a subset of the edges of G_A.

The input to the algorithm is an n-by-n SPDDD matrix A, with $2m$ off-diagonal nonzeros and a parameter t. We begin by finding a rooted maximum-weight spanning tree T in G_A. We decompose T into a set of k connected subgraphs $V_1, V_2, \ldots V_k$ such that each V_i has between n/t and $(dn/t)+1$ vertices, where d is the maximal number of children that vertices in T have. We form G_M by adding to T the heaviest edge between V_i and V_j for all i and j. We add nothing if there are no edges between V_i and V_j or if the heaviest edge is already in T. The weight of an edge (i, j) in G_M is the weight of the same edge (i, j) in G_A. We assign weight to a self loop (i, i), which corresponds to a diagonal element $M_{i,i}$, so that the row sums in M and in A are identical. The preconditioner M is the matrix whose underlying graph is G_M.

We denote by M_t the Vaidya preconditioner constructed with the parameter t. We have $M_n = A$. The preconditioner M_1 consists solely of a maximum-weight spanning tree with no added edges. Bern et al. [6] show that the condition number of the preconditioner M_1 is $O(mn)$.

In general, Bern et al. show that the condition number of Vaidya's preconditioner is $O(n^2/k^2)$, where k is the number of subgraphs that T is actually split into. They also analyze the cost of factoring M when G_A is a bounded-degree graph or a planar graph. The results of these analyses are summarized in the Introduction; we omit further details.

2.2. Implementation Details. There are two implementation issues that must be addressed in the construction of Vaidya's preconditioners. One is the choice of the maximum-spanning-tree algorithm and the second is the splitting of the tree into subtrees.

We use Prim's algorithm to find the maximum-weight spanning tree T [25] because it is fast and because it returns a rooted tree. The root r, which we choose randomly, is an input to Prim's algorithm. (Most textbooks on algorithms describe Prim's algorithm, as well as other maximum spanning tree algorithms; see, for example, [10, Chapter 24].) Prim's algorithm returns a rooted tree represented by an array π. The integer $\pi[i]$ represents the parent of the vertex i. We use π to create two length-n integer arrays that allow us to quickly enumerate the children of a vertex. Figure 2.1 shows a matrix, its graph and the maximum spanning tree.

We now present a recursive procedure called TREEPARTITION, which is specified in Figure 2.2, that decomposes T into a set of connected subtrees. The number of vertices in each subtree is between n/t and $(dn/t)+1$, except for the subtree containing the root, which might be smaller. Ideally, we would have liked to split T into exactly t connected subtrees of nearly equal size, but we are not aware of an algorithm that does so. TREEPARTITION uses a global array s of n integers, where s_i is initialized before the first call to TREEPARTITION to be the number of vertices in the subtree rooted at i. The initial call to TREEPARTITION passes the root of T as an argument.

THEOREM 2.1. TREEPARTITION(i) *splits the subtree rooted at i into connected subtrees whose size is between* n/t *and* $(dn/t) + 1$*, except perhaps for the subtree that contains i, which may be smaller (but not larger).*

Proof. We prove by induction on the height of the tree a slightly stronger statement. Namely, that when TREEPARTITION returns, the tree is split appropriately and that the number of vertices in the subtree rooted at i is s_i. The claim is obviously true for leaves since s_i

ETNA
Kent State University
etna@mcs.kent.edu

$$A = \begin{bmatrix} 12 & -2 & -5 & & -4 & & & \\ -2 & 7 & -5 & & & -6 & & \\ -5 & -5 & 17 & -6 & -1 & & & \\ & & -6 & 8 & & -2 & & \\ -4 & & -1 & & 12 & -3 & -4 & \\ & -6 & & -2 & -3 & 16 & -5 & \\ & & & & -4 & -5 & 15 & -6 \\ & & & & & & -6 & 6 \end{bmatrix}$$

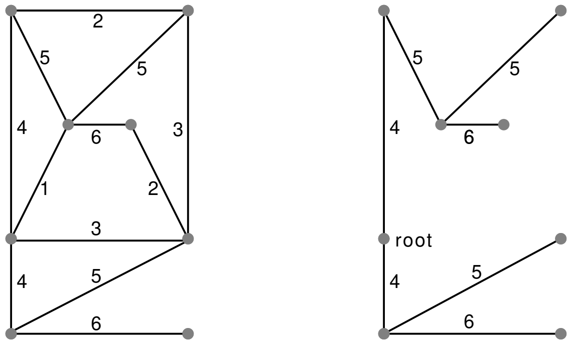


FIG. 2.1. *An SPDDD matrix A (top), the graph G_A of A (bottom left), and its rooted maximum spanning tree T (bottom right). The vertices are ordered top to bottom, left to right. The choice of a root for T is arbitrary. Self loops, which correspond to diagonal matrix elements, are not shown.*

is set to 1. Suppose that the height of the tree rooted at i is h and that the claim is true for $h' < h$. Clearly, the height of the trees rooted at a child j of i is smaller than h. For each child j, if we call TREEPARTITION(j), then by induction all the subtrees that are formed inside the recursive call have the right sizes and s_j is set to the size of the subtree that remains rooted at j. Therefore, when we test whether s_j is greater or equal to n/t, s_j is correct and is at most $(dn/t) + 1$. If $s_j \geq n/t$, then the subtree rooted at j has a valid size so we can form a new subtree. Otherwise, it is too small and we leave it connected to i and add its size to s_i. When TREEPARTITION(i) terminates, s_i is therefore correct and at most $(dn/t) + 1$. The connectedness of the subtrees follows from a similar inductive claim. $\square$

Making the recursive calls only when $s_j > (dn/t) + 1$ is correct, but our experiments indicate that this modification degrades the preconditioner. It appears that if we make the recursive call only when $s_j > (dn/t) + 1$, the subtrees tend to significantly differ in size. If we make the recursive call only when $s_j > n/t + 1$, then the algorithm tends to generate subtrees whose size is close to the lower bound n/t, so they are more uniform in size. We have no theoretical analysis of this phenomenon.

In addition, making a recursive call whenever the subtree is large enough, larger than $(n/t) + 1$, allows the algorithm to partition the graph into as many as n subgraphs. On the other hand, making the recursive calls only when $s_j > (dn/t) + 1$ typically limits the number of subgraphs that the graph can be partitioned into, which in turn limits the condition number that we can reach. Hence, making a recursive call whenever possible gives us more control over fill and condition number than making the recursive call only when necessary.

Adding the heaviest edge in G_A between every pair of subtrees is trivial; we omit the details.

```
TREEPARTITION(vertex i, integer array s)
    % s_i = number of vertices in the subtree initially rooted at i.
    % When this function returns, s_i may shrink if we partition the
    % subtree rooted at i.
    % We set s_i to 1 to count i itself, and later add the sizes of
    % subtrees rooted at the children that remain connected to it
    s_i ← 1
    for each child j of i
        % We first ensure that the subtree rooted at j
        % is not too large, by partitioning it recursively if possible.
        if (s_j > n/t + 1)
            TREEPARTITION(j)
            % Now 1 ≤ s_j ≤ (dn/t) + 1 is not too large.
            % Vertex j is still connected to i.
        % We now decide whether to disconnect the subtree
        % rooted at j or keep it connected to i. We may decide
        % to disconnect it even if TREEPARTITION(j) was called
        % in the first conditional.
        if (s_j ≥ n/t)
            form a new subtree rooted at j
            disconnect j from i
        else
            s_i ← s_i + s_j
    % s_i = number of vertices in the subtree rooted at i.
```

FIG. 2.2. *The algorithm that we use to decompose the maximum spanning tree. The code splits the tree T, which is stored in a global data structure. The code uses a global integer array s. Lines starting with a % sign are comments.*

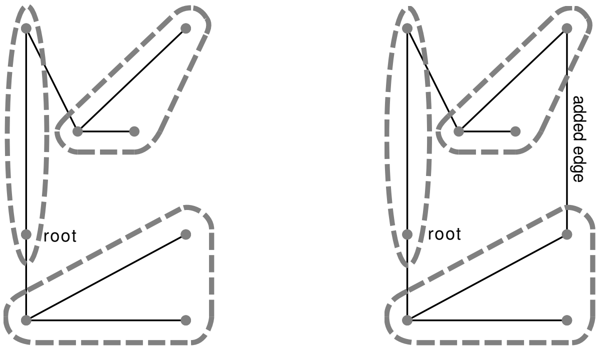
3. Experimental Results.

3.1. Methodology. Both Vaidya's preconditioners and drop-tolerance incomplete-Cholesky preconditioners accept a parameter that indirectly affects the sparsity of the preconditioner. In Vaidya's preconditioner the parameter t affects the number of subgraphs that the tree is split into, whereas in the incomplete-Cholesky preconditioner the drop-tolerance parameter determines which fill elements are dropped from the factors.

We normally compare preconditioners with similar amounts of fill in the factor L of the preconditioner. The amount of memory required to represent a factor is proportional to the number of nonzeros in the factor. In addition, the number of floating-point operations required to apply a preconditioner is about twice the number η_L of nonzeros in the factor L of the preconditioner. Hence, the number of nonzeros is a strong determinant of the running time of each iteration in an iterative solver (parallelism and cache effects also influence the running time but we ignore them in this paper). Finally, computing the factors usually takes more time when they are denser, although the dependence is somewhat complex. The number of operations in a complete symmetric factorization is proportional to the sum of squares of nonzero counts for each column.

It may occur that the optimal amount of fill for Vaidya's preconditioners is different from the optimal amount of fill for IC or MIC preconditioners. To address this issue, we provide in most cases sufficient data to allow the reader to determine how different amounts of fill affect each preconditioner, usually in the form of graphs that plot total solution time as a function

ETNA
Kent State University
etna@mcs.kent.edu



$$M = \begin{bmatrix} 10 & & -5 & & -4 & & & \\ & 11 & -5 & & & -6 & & \\ -5 & -5 & 16 & -6 & & & & \\ & & -6 & 6 & & & & \\ -4 & & & & 8 & & -4 & \\ & -6 & & & & 11 & -5 & \\ & & & & -4 & -5 & 15 & -6 \\ & & & & & & -6 & 6 \end{bmatrix}$$

FIG. 2.3. *The tree from Figure 2.1 partitioned (top left), and the graph G_M of the Vaidya preconditioner (top right). The tree was partitioned using* TREEPARTITION *with $t = 4$. Two edges that connect the 3 subtrees are already in the tree; to complete the preconditioner we added one more edge, the heaviest between the subtrees that are not connected by a tree edge. The matrix on the bottom is the preconditioner M itself.*

of fill.

3.2. Experimental Setup. The experiments were conducted on a dual-processor 600 MHz Pentium III computer with 2 GBytes of main memory. We only use one processor. The computer runs the Linux operating system, which on this machine can actually allocate about 1900 MBytes to a process in a single `malloc`. The code is written in C and was compiled using `gcc` version 2.95.2 with the options `-O3`.

We use METIS version 4.0 [17, 18] or Joseph Liu's GENMMD code to find fill-reducing orderings. With METIS, we use the procedure `METIS_NodeND` with default options to find the ordering. GENMMD has a single user-specified parameter, `delta`, which we set to 1. In Vaidya's preconditioners, the main difference between METIS and GENMMD is that METIS is more expensive but produces less fill on large problems; the convergence behavior is independent of the ordering. Since our experiments use matrices whose graphs are regular meshes in 2 and 3 dimensions, we also run IC and MIC with the natural ordering of the mesh. Unrelaxed MIC preconditioners break down with METIS and GENMMD orderings. We expect that for unstructured meshes, envelope minimizing orderings such as Cuthill-McKee [11] or Sloan [19] would produce results similar to natural orderings of regular meshes [13].

The experiments use two sparse Cholesky factorization algorithms that we have implemented. One code is a supernodal multifrontal sparse Cholesky code [12, 20]. This code can only perform complete factorizations, so we use it only to factor Vaidya's preconditioners. The other code is a column-oriented left-looking sparse Cholesky code. The code is efficient in the sense that its running time is proportional to the number of floating-point operations that it performs. It can perform complete, no-fill incomplete (sometimes known as IC(0) or ICCG(0) [21]), and drop-tolerance incomplete factorization. When performing incomplete

ETNA
Kent State University
etna@mcs.kent.edu

factorizations, the code can modify the diagonal to ensure that the row sums of LL^T are equal to those of A [15] or it can use a relaxed modification, which is more robust [2, 4, 5, 28]. The performance of this code is similar to the performance of other drop-tolerance incomplete-Cholesky codes, but it is slower than the multifrontal code unless L remains very sparse.

The iterative solver that we use is preconditioned conjugate gradients (see, for example, [3, 14]).

3.3. Test Problems. The matrices that we use for our experimental analysis are discretizations of elliptic PDEs on regular 2- and 3-dimensional meshes. Most of the matrices arise from finite-differences discretizations of the equation

$$c_x \frac{\partial^2 u}{\partial x^2} + c_y \frac{\partial^2 u}{\partial y^2} = f \qquad \text{in } \Omega =]0,1[\times]0,1[$$

with either Dirichlet or Neumann boundary conditions. We solve isotropic problems ($c_x = c_y = 1$) and anisotropic problems in which either $c_x = 100$ and $c_y = 1$ or vice versa. We also solve similar problems in 3D.

The remaining matrices arise from problems with discontinuous coefficients in 3D. These problems have the form

$$c\frac{\partial^2 u}{\partial x^2} + c\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f \qquad \text{in } \Omega =]0,1[\times]0,1[\times]0,\ell[$$

with Neumann boundary conditions and with $\ell \geq 1$. We set

$$c(x,y,z) = \begin{cases} \alpha & x \leq \frac{1}{8} \text{ or } y \leq \frac{1}{8} \\ 1 & \text{otherwise,} \end{cases}$$

where α, the jump in the coefficients, is a parameter. We generate the grids so that grid lines coincide with the discontinuity and so that the distance between grid lines is the same in all three directions (that is, a 32-by-32-by-200 grid corresponds to $\ell = 6.25$).

Our aim in choosing these problems is mainly to highlight different aspects of the behavior of Vaidya's preconditioners. In particular, these problems are not necessarily common in applications. All of the resulting matrices are diagonally-dominant Stieltjes matrices.

We use a five-point discretization in 2D and a seven-point discretization in 3D, which lead to a pentadiagonal matrix when a 2D mesh is ordered row-by row (the so-called natural order) or to a septadiagonal matrix in 3D.

We have been unable to find large unstructured SPDDD matrices (over a million unknowns, say) in matrix collections, such as MatrixMarket and Tim Davis's collection.

The exact solution u in the experiments reported below is a random vector whose elements are uniformly and independently distributed in $[0,1]$. We also performed experiments on the 3D problems with smooth solutions of the form

$$u(x,y,z) = (xyz(1-x)(1-y)(\ell - z))^2 \, e^{x^2yz} \, .$$

We have obtained similar results to the ones reported here, so we omit these from the paper.

We usually solve these systems quite accurately, reducing the 2-norm of the residual by 10^{15}. Although this level of accuracy is often unusual for iterative solvers, it makes sense in this paper. First, some of the problems that we solve are ill conditioned, so dropping the residual by only 10^8, say, may produce a solution with essentially no correct bits. Second, stopping the iterations only when the residual is very small allows us to observe various phenomena, especially with IC and MIC preconditioners, that do not occur when the residual

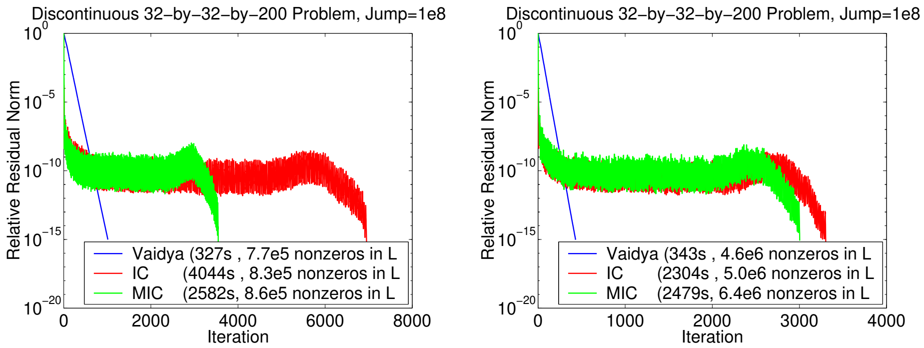


FIG. 3.1. *The convergence of preconditioned CG on the discontinuous problem on a* 32*-by-*32*-by-*200 *grid. The graphs also show the total solution times (in the legends). The magnitude of the jump in the coefficients is* 10^8*. Each graph shows three preconditioners, Vaidya, IC, and relaxed MIC with approximately the same number of nonzeros in L: between* 7.7×10^5 *and* 8.6×10^5 *on the left, and between* 4.6×10^6 *and* 6.4×10^6 *on the right. We used the natural ordering of the grid for IC and MIC; using GENMMD for MIC on this problem leads to qualitatively similar results with similar numbers of iterations and similar solution times; in particular, using GENMMD does not fix the stagnation at all. We used GENMMD for Vaidya.*

is relatively large. Third, the experiments clearly show that this level of accuracy is attainable. Fourth, we plot the drop in the residual for many of the experiments, so the reader can judge for herself/himself how different accuracy requirements affect the performance of the solvers. In some experiments on relatively well-conditioned matrices we reduce the residual only by a factor of 10^8; this is indicated clearly in the text and the figure captions.

We deal with singular systems (Neumann boundary conditions) by adding the equation $u_1 = 0$ to the first equation. This diagonal perturbation ensures that the coefficient matrix is an SPDDD, and in particular nonsingular. This method of removing the singularity does not seem to cause any problem to Vaidya's preconditioners. We have also performed IC and MIC experiments in which we dealt with the singularity by orthogonalizing the right-hand-side against the constant vector and by orthogonalizing the residuals in every iteration against the constant vector. We have not detected any significant differences between these results an the results of the diagonal perturbation method, so these results are not reported here. For a more thorough treatment of the iterative solution of consistent singular systems arising from Neumann boundary conditions, see the recent paper by Bochev and Lehoucq [7].

3.4. Experimental Results: 3D Problems with Discontinuous Coefficients. Figures 3.1 and 3.2 show that Vaidya's preconditioners work well on ill-conditioned problems that causes difficulties to IC and MIC preconditioners. In these runs, we reduced the residual by a factor of 10^{15}. Since A is ill-conditioned, a small residual is required in order to achieve a reasonably small forward error. The figures clearly show that it is possible to solve these systems to this level of accuracy. Figure 3.1 shows that Vaidya's preconditioners converge monotonically and at a nearly constant rate on a discretization of a PDE with discontinuous coefficients, whereas IC and relaxed MIC preconditioners essentially stagnate for thousands of iterations. A comparison of the graph on the left with the graph on the right shows that allowing the preconditioners to fill more improves the convergence of both families of preconditioners, but does not change their behavior in a fundamental way. Figure 3.2 shows that the stagnation of relaxed MIC preconditioners is caused by the large jump in the coefficients. When there is no jump (constant continuous coefficients), relaxed MIC preconditioners converge quickly without any stagnation. When the jump is 10^4, relaxed MIC preconditioners stagnate; when the jump is 10^8 it stagnates for much longer. Vaidya's preconditioners are

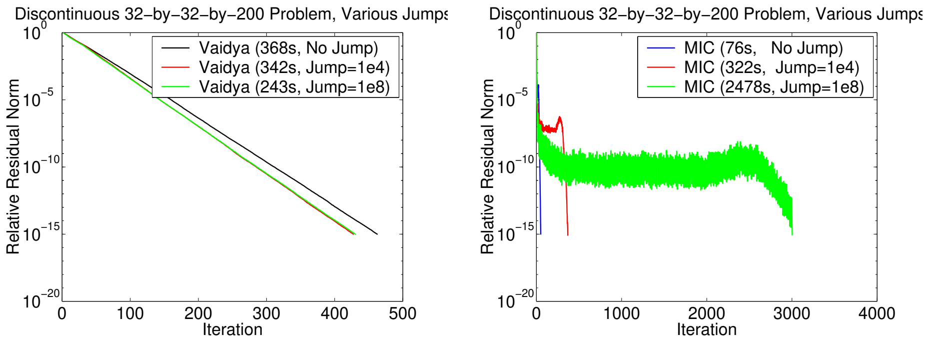


FIG. 3.2. *The convergence of preconditioned CG on the discontinuous problem on a* 32*-by-*32*-by-*200 *grid as a function of the jump in the coefficients. The graphs also show the total solution times (in the legends). The graph of the left shows the convergence of Vaidya with no jump, a jump of* 10^4*, and a jump of* 10^8*; these three preconditioners all have* 4.6×10^6 *nonzeros in L. The graph on the right shows the convergence of relaxed MIC with the same three jumps; these preconditioners have between* 6.4×10^6 *and* 8.8×10^6 *nonzeros in L. The behavior of IC, which is not shown, is qualitatively similar to that of relaxed MIC, although convergence is slower. We again used the natural ordering for MIC and GENMMD for Vaidya (using GENMMD for MIC does not change its performance significantly).*

almost insensitive to the size of the jump. In fact, when the jump is large, they converge slightly faster than when there is no or almost no jump.

Figure 3.2 also shows that on relatively well-conditioned problems (no jump), relaxed MIC outperforms Vaidya's preconditioners. With a medium size jump (10^4), the performance of relaxed MIC and of Vaidya's preconditioners are roughly comparable, but with a large jump, Vaidya's preconditioners leads to much faster solution times.

These results do not imply that IC and MIC preconditioners stagnate on matrices arising from problems with discontinuous in general, only that they stagnate on this particular problem. In fact, we experimented with various discontinuous problems, and on many of them IC and MIC preconditioners do not stagnate. The behavior of Vaidya's preconditioners is independent of the details of the discontinuity. Hence, we cannot conclude from this experiment that IC and MIC preconditioners perform poorly on discontinuous problems, but we can conclude that on some problems that cause IC and MIC preconditioners difficulties, Vaidya's preconditioners work well.

3.5. Experimental Results: Eigenvalue Distributions. The nearly constant convergence rate of Vaidya's preconditioners, evident in both Figure 3.1 and Figure 3.2, suggests that the eigenvalues of the preconditioned system are probably distributed evenly throughout the spectrum. In general, this is not a desirable property: clustered eigenvalues would lead to faster convergence, at least on well-conditioned problems. We have observed the same behavior in all of our experiments. Figure 3.3 explores this issue further by presenting the convergence and eigenvalues of both Vaidya and relaxed-MIC preconditioners on two problems that are small enough that we can compute the eigenvalues numerically. One problem is a difficult 3D problem with discontinuous coefficient that clearly displays the same qualitative convergence behavior that we have seen in Figures 3.1 and 3.2. Figure 3.4 plots the eigenvalues of the unpreconditioned coefficient matrix of this ill-conditioned problem. The other is a 2D problem with constant coefficients on which Vaidya and relaxed MIC preconditioners perform similarly, as shown both here and in Section 3.7.

On both problems, the eigenvalues of the Vaidya-preconditioned operator range from 1 to the condition number of the preconditioned system, which are approximately 288 (3D problem) and 83 (2D problem). The theory of Vaidya's preconditioners predicts that the eigenvalues are bounded from below by 1. There is a fairly large cluster of eigenvalues near 1 (478 eigenvalues in the 2D problem) and the rest are larger. The eigenvalues are not distributed uniformly but they have no significant clusters either.

The Relaxed-MIC-preconditioned operators have a qualitatively different eigenvalue distribution. The spectrum contains a few eigenvalues below 1 (29 in the 2D problem, about 316 in the 3D problem). It has a larger cluster at 1 than the Vaidya preconditioner (1201 eigenvalues in the 2D problem). Above 1, the spectrum is much more clustered than Vaidya's, especially in the 3D problem, and the large eigenvalue is smaller than Vaidya's. The main difference between the difficult 3D problem, on which MIC preconditioners clearly stagnate, and the easy 2D problem on which it converges monotonically, is the behavior of the small eigenvalues. In the easy 2D problem, the smallest is 0.19, not particularly small; on the difficult 3D problem, some are tiny, the smallest being 8.3×10^{-10}. On the 3D problem the MIC-preconditioned oparator has a large condition number, around 1.6×10^{10}, versus only 288 for Vaidya's preconditioner. It seems likely, however, that it is not the large condition number of MIC preconditioners that is causing the stagnation, but numerical issues. Although the MIC-preconditioned operator has a large condition number, its eigenvalues are well clustered. It appears that what is causing the problem is the magnitude of the small eigenvalues. These small eigenvalues may cause loss of conjugacy between the direction vectors, or they may cause an error in the direction of the corresponding eigenvectors to reappear in the approximate solution after they have been filtered out. In order to explore this issue further, we repeated the 3D experiment in Matlab using two other iterative solvers: MINRES and GMRES without restarts. The two solvers showed similar convergence behavior, and neither converged monotonically (that is, the 2-norm of the residuals oscillated). This suggests the loss of orthogonality is not the problem. This issue merits further investigation, beyond the scope of this paper.

Clearly, the distribution of the eigenvalues in Vaidya-preconditioned operators does not appear to contain extremely small or large eigenvalues, which helps explain its robustness. On the other hand, the distribution also does not appear to contain significant clusters besides the one at 1, which helps explain why its basic convergence rate is not particularly fast: the worst-case Chebychev polynomial approximation bound, which depends only on the extreme eigenvalues, is probably fairly tight. The eigenvalue distribution of MIC preconditioners (and probably IC preconditioners) tends to be more clustered, which helps explain its effectiveness on many problems, but it may have a few tiny outliers that cause difficulties.

3.6. Experimental Results: 3D problems with Constant Coefficients. Figure 3.5 shows that on a well-conditioned 3D problem (Neumann boundary conditions and constant coefficients), MIC preconditioners beat Vaidya's preconditioners no matter how much fill we allow. Figure 3.2 reports similar performance on a differently-shaped region when there is no jump. Table 3.1, which reports additional data on the same experiment, helps to explain this phenomenon. The data in the table shows that Vaidya's preconditioners perform poorly due to two main reasons:

1. They converge much slower than MIC preconditioners given similar amounts of fill. For example, a MIC preconditioner with 2.0×10^7 nonzeros in L converges in 88 iterations, but Vaidya's preconditioner with more fill converges in 674 iterations.
2. They take longer to factor than MIC preconditioners given similar amounts of fill. This is not due to some inefficiency in the factorization code, but due to a poorer distribution of nonzeros in the columns of L, which requires more floating-point operations to factor. (The number of flops to factor a sparse matrix is proportional to the sum of the squares of the nonzeros in each column of L, and in Vaidya's

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 3.1
The performance of Vaidya and MIC preconditioners on a 3D 100*-by-*100*-by-*100 *isotropic problem with Neumann boundary conditions. The table shows the parameter (drop tolerance or the number t of subgraphs) used to construct each preconditioners, the number of nonzeros in L, the preconditioner-creation time T_c, the ordering time T_o, the factorization time T_f, the iterations time T_s, the total solution time T_t, and the number of iterations to reduce the norm of the residual by 10^{15}. In this experiment GENMMD leads to even slower times for Vaidya.*

Precond	Param	Ordering	nnz(L)	T_c	T_o	T_f	T_s	T_t	# its
MIC	0.1	GENMMD	4.0e6	—	58.7	7.4	1121.9	1187.0	669
MIC	0.01	GENMMD	1.1e7	—	58.7	24.2	349.7	431.6	135
MIC	0.003	GENMMD	2.0e7	—	58.7	55.1	305.0	418.2	88
MIC	0.001	GENMMD	3.2e7	—	58.7	138.5	293.9	490.2	63
Vaidya	1000	METIS	3.3e6	8.8	51.2	26.4	3274.4	3360.8	2460
Vaidya	3000	METIS	5.2e6	8.9	53.7	97.2	2092.7	2252.5	1381
Vaidya	10000	METIS	1.0e7	9.2	55.9	360.9	1833.0	2258.1	900
Vaidya	30000	METIS	2.2e7	9.6	58.6	825.3	2083.5	2976.5	674

preconditioners on 3D problems the sum of squares is much larger than the sum of squares in IC and MIC preconditioners, even when the sums themselves are similar.)

Due to these two reasons, a relatively sparse Vaidya preconditioner takes too long to converge, and a relatively dense one takes too long to factor without reducing the number of iterations sufficiently to render it efficient.

3.7. Experimental Results: 2D Problems with Constant Coefficients. The situation is quite different for two dimensional problems. Figure 3.6 shows that the performance of Vaidya's preconditioners is comparable to relaxed-MIC preconditioners with similar amounts of fill. There are small differences in performance, but they are not very significant. Without diagonal modification, IC preconditioners perform more poorly than both relaxed-MIC and Vaidya preconditioners. The figure shows that in spite of theoretical results that suggest that both MIC and Vaidya's preconditioners should outperform direct solvers asymptotically, direct solvers are very effective on 2D problems. Although the asymptotic arithmetic complexity of a direct solver on an n-node 2D grid is $\Theta(n^{1.5})$, it beats in practice the asymptotically more efficient Vaidya and MIC solvers. The reason for this behavior is most likely the good cache behavior and the low overhead of the multifrontal factorization code that we are using. In 3D the situation is completely different since direct solvers incur a huge amount of fill. (The situation is also different in 2D when the required reduction in the norm of the residual is not large; in this case, iterative solvers run faster whereas direct solvers cannot exploit the lax accuracy requirement.) The graph also shows the difference between the convergence behavior of Vaidya and that of IC and relaxed-MIC preconditioners on this problem, when the factors of all three preconditioners have approximately 1.6×10^7 nonzeros. Vaidya's preconditioners exhibit the same nearly-constant convergence rate that we have also seen in Figures 3.1 and 3.2. IC and relaxed MIC preconditioners converge essentially monotonically, which was not the case in Figures 3.1 and 3.2, but their convergence is more erratic than that of Vaidya's preconditioners: the first few iterations are very effective, they then stagnate for a while, and then converge at an approximately constant rate, but slower than Vaidya's (and slower for IC than for relaxed MIC).

Figure 3.7 and Table 3.2 demonstrate that Vaidya's preconditioners scale well on 2D problems as a function of the mesh size. The data shows that Vaidya's preconditioners scale well with the mesh size. The number of iterations grows as the mesh grows, but only slowly.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 3.2
The number of iterations for Vaidya's preconditioners with $10n$ nonzeros in L on isotropic 2D problems with Neumann boundary conditions. The reduction in the residual is by a factor of 10^8. The data shows that Vaidya's preconditioners scale well with problem size. The numbers of iterations with Dirichlet boundary conditions are the same, except that 51 rather than 56 iterations were required on grid size 700.

Grid Size	300	500	700	900	1100	1300	1500
Iterations	41	44	56	53	63	63	64

Note that the experiment uses preconditioners with approximately $10n$ nonzeros in L, so constructing them and applying them is roughly linear in the size of the mesh. In other words, these Vaidya preconditioners combine linear scaling of the work per iteration with a slow growth in the number of iterations, a highly desirable behavior. The data also shows that Vaidya's preconditioners are insensitive to the difference between Neumann and Dirichlet boundary conditions.

Figure 3.7 also shows that IC and MIC preconditioners are sensitive to the boundary conditions. The experiments show that at this level of fill in the preconditioners, the number of iterations that Vaidya's preconditioners perform scales similarly to MIC's scaling on Dirichlet boundary conditions.

3.8. Experimental Results: Anisotropic 2D Problems. Vaidya's preconditioners are unaffected by the original ordering of the matrix and are capable of automatically exploiting numerical features. On anisotropic problems, Vaidya's preconditioners are almost completely unaffected by whether the direction of strong influence is the x or the y direction, as shown in Figure 3.8. The construction of Vaidya's preconditioners starts with a maximum spanning tree, which always include all the edges (nonzeros) along the direction of strong influence. They are, therefore, capable of exploiting the fact that there is a direction of strong influence and they lead to faster convergence than on an isotropic problem. IC preconditioners, on the other hand, are sensitive to the interaction between the elimination ordering and the direction of strong influences. Figure 3.8 shows that naturally-ordered relaxed MIC is much more effective for one direction of strong influence than for the other when the matrix ordering is fixed. When we use a minimum-degree ordering for relaxed MIC, its performance becomes independent of the direction of strong influence.

Figure 3.8 also shows that the performance of Vaidya's preconditioners on this problem is better or similar to the performance of relaxed MIC preconditioners, especially when the preconditioners are sparse.

Similar experiments with unmodified IC preconditioners showed that their performance is worse than that of relaxed MIC preconditioners, so it is not shown in these graphs.

4. Conclusions . Our main findings from this experimental study are as follows.

- Vaidya's preconditioners converge at an almost constant rate on a variety of problems. This behavior is very different from that of incomplete Cholesky preconditioners, which have periods of rapid convergence, periods of stagnation, and periods of roughly constant-rate convergence.
- As predicted by the theoretical analysis of Vaidya's preconditioners, they are sensitive only to the nonzero structure of the coefficient matrix and not to the values of its entries, as long as the matrices remain SPDDD matrices. Again, the behavior is different from that of incomplete Cholesky preconditioners, which are sensitive to the boundary condition, to anisotropy, and to discontinuities.
- Vaidya's preconditioners deliver comparable performance to that of incomplete Cholesky on a wide variety of two-dimensional problems. Although we have not

ETNA
Kent State University
etna@mcs.kent.edu

tested them against state-of-the-art IC preconditioners or against many variants, we have tested them against relaxed drop-tolerance MIC preconditioners, which are widely used in practice, and we have found them to be competitive.

- Vaidya's preconditioners deliver poorer performance than IC preconditioners with similar amounts of fill on some 3D problems, but dramatically better performance on other 3D problems. As a rule of thumb, if IC preconditioners performs poorly due to long stagnation, then Vaidya's preconditioners may perform better because it is unlikely to stagnate.

The nearly constant convergence rate of Vaidya's preconditioners have several implications. In general, this behavior is not desirable, since it suggests that the eigenvalues of the preconditioned system are evenly distributed throughout its spectrum. This implies that the worst-case Chebychev polynomial approximation bound, which depends only on the extreme eigenvalues, is probably tight. An eigenvalue distribution with clusters is more desirable, but Vaidya's preconditioners do not seem to produce clustered eigenvalues. On the other hand, the eigenvalue distribution of Vaidya's preconditioners seems to avoid stagnation. Therefore, Vaidya's preconditioners seem more appropriate for "difficult problems" on which other preconditioners lead to non-convergence or to long stagnation. Vaidya's preconditioners are less appropriate for situations in which low accuracy is acceptable, since other preconditioners, such as IC, may attain the required accuracy before they stagnate at a higher initial convergence rate than Vaidya's.

Figure 4.1 shows one way in which the insight concerning the different convergence behavior of Vaidya and IC preconditioners can be used. The figure shows that on a problem on which IC preconditioners stagnate, we can exploit the rapid initial convergence rate of IC preconditioners by starting the iterations with a cheap IC(0) preconditioner and then switching to Vaidya. In principle, the loss of information incurred by restarting conjugate gradient may cause the solver to stagnate, but we see that in this case we do obtain a significant savings in running time over a plain Vaidya preconditioner (and over IC preconditioners, which stagnate on this problem).

Another way to exploit the nearly constant convergence rate is in time-to-completion predictions. Although progress bars, like the ones that browsers display to indicate the status of a file download, are not commonly used in numerical software, they could be quite useful to users. A user often has no idea whether an action that he or she initiated (in MATLAB, say) is going to run for an hour or two minutes. A progress bar can help the user determine whether they wish to wait or to abort a long-running computation. In many iterative solvers predicting the solution time is quite difficult because of the variable convergence rate. But with Vaidya's preconditioners the completion time can probably be estimated quite accurately after a few iterations. The time to complete the factorization phase can also be estimated quite accurately once the elimination tree and the nonzero counts for the columns (which indicate flop counts) have been computed. The ordering time is more difficult to estimate but is often insignificant.

Our codes are publicly available in a package called TAUCS.[2] The package includes iterative solvers, codes that construct Vaidya's preconditioners (both for SPDDD matrices and for more general SDD matrices), an incomplete Cholesky factorization code, a multifrontal Cholesky factorization code, interfaces to METIS and other matrix ordering codes, matrix generation and I/O routines. The code that constructs Vaidya's preconditioners can be used, of course, with other iterative solver codes and other ordering and factorization codes. Our aim in supplying these codes is both to allow Vaidya's preconditioners to be integrated into linear solver libraries and into applications, and to facilitate further research on this class of

[2] `http://www.tau.ac.il/~stoledo/taucs`.

preconditioners.

We believe that Vaidya's preconditioners are worthy of further investigation. Are there effective Vaidya-like preconditioners for 3D problems? Are there specific maximum spanning trees for problems with constant coefficients that lead to faster convergence? Can we benefit from starting from near-maximal spanning trees with better combinatorial structure? How can we extend Vaidya's preconditioners to more general classes of matrices?

Two ideas that have appeared in the literature suggest how Vaidya's preconditioners can be improved, especially on 3D problems where our study indicates that they are often slow. In his PhD thesis, Joshi suggests a different edge-dropping algorithm for matrices arising from regular finite-differences discretization of Poisson problems with constant coefficients [16]. Although preconditioners limited to regular problems with constant coefficients are not particularly useful given the state of the art in multigrid and domain decomposition, edge-dropping methods inspired by Joshi's ideas may prove useful for more challenging problems. Another idea concerns recursion. This idea was mentioned by Vaidya [27] and further explored by Joshi [16], Reif [26], and Boman [8]. The main idea is to form a Vaidya preconditioner M but to avoid factoring it completely. Instead, we eliminate some of the rows and columns in M and form the Schur complement (the reduced trailing submatrix). Instead of factoring the Schur complement, we form a Vaidya preconditioner for the Schur complement. Each preconditioning operation in the outer iterative linear solver now requires solving a Schur-complement problem iteratively. Our code can construct and use such preconditioners, but additional research is required to determine how effective they are. (We performed limited experiments with recursive Vaidya preconditioners and have not found problems on which they were more effective than plain Vaidya preconditioners.)

Acknowledgements.. Thanks to Bruce Hendrickson and to Erik Boman for numerous discussions and suggestions. Thanks to Rich Lehoucq for helpful comments and for pointing us to [24]. Thanks to Henk van der Vorst and to Eldad Haber for helpful comments. Thanks to Cleve Ashcraft for helping us resolve problems with SPOOLES, which we used in early experiments not reported here. Thanks to the two anonymous referees for their helpful comments.

REFERENCES

[1] M. A. AJIZ AND A. JENNINGS, *A robust incomplete Choleski-conjugate gradient algorithm*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 949–966.

[2] C. ASHCRAFT AND R. GRIMES, *On vectorizing incomplete factorizations and SSOR preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), no. 1, pp. 122–151.

[3] O. AXELSSON, *Iterative Solution Methods*, Cambridge University Press, 1994.

[4] O. AXELSSON AND G. LINDSKOG, *On the eigenvalue distribution of a class of preconditioning matrices*, Numer. Math., 48 (1986), pp. 479–498.

[5] ———, *On the rate of convergence of the preconditioned conjugate gradient method*, Numer. Math., 48 (1986), pp. 499–523.

[6] M. BERN, J. R. GILBERT, B. HENDRICKSON, N. NGUYEN, AND S. TOLEDO, *Support-graph preconditioners*, Tech. Rep., School of Computer Science, Tel-Aviv University, 2001.

[7] P. BOCHEV AND R. B. LEHOUCQ, *On finite element solution of the pure Neumann problem*, Tech. Rep. SAND2001-0733J, Sandia National Laboratories, 2001.

[8] E. G. BOMAN, *A note on recursive Vaidya preconditioners*. Unpublished manuscript, Feb. 2001.

[9] E. G. BOMAN, D. CHEN, B. HENDRICKSON, AND S. TOLEDO, *Maximum-weight-basis preconditioners*. To appear in *Numer. Linear Algebra Appl.*, 29 pages, June 2001.

[10] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press and McGraw-Hill, 1990.

[11] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 24th National Conference of the Association for Computing Machinery, 1969, pp. 157–172.

[12] I. DUFF AND J. REID, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[13] I. S. DUFF AND G. MEURANT, *The effect of ordering on preconditioned conjugate gradient*, BIT, 29 (1989), pp. 635–657.

[14] G. H. GOLUB AND C. F. V. LOAN, *Matrix Computations*, Johns Hopkins University Press, 3rd ed., 1996.

[15] I. GUSTAFSSON, *A class of first-order factorization methods*, BIT, 18 (1978), pp. 142–156.

[16] A. JOSHI, *Topics in Optimization and Sparse Linear Systems*, PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1997.

[17] G. KARYPIS AND V. KUMAR, *Multilevel k-way partitioning scheme for irregular graphs*, J. Parallel Distrib. Comput., 48 (1998), pp. 96–129.

[18] ———, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, J. Parallel Distrib. Comput., 48 (1998), pp. 71–85.

[19] G. KUMFERT AND A. POTHEN, *Two improved algorithms for reducing the envelope size and wavefront of sparse matrices*, BIT, 18 (1997), pp. 559–590.

[20] J. W. H. LIU, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.

[21] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[22] Y. NOTAY, *DRIC: A dynamic version of the RIC method*, Numer. Linear Algebra with Appl., 1 (1994), pp. 511–532.

[23] E. L. POOLE, M. HEROUX, P. VAIDYA, AND A. JOSHI, *Performance of iterative methods in ANSYS on Cray parallel/vector supercomputers*, Comput. Systems Engrg., 3 (1995), pp. 251–259.

[24] G. POOLE, Y.-C. LIU, AND J. MANDEL, *Advancing analysis capabilities in ANSYS through solver technology*, in Proceedings of the 10th Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, April 2001.

[25] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Technical Journal, 36 (1957), pp. 1389–1401.

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

[27] P. M. VAIDYA, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners*. Unpublished manuscript. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis.

[28] H. VAN DER VORST, *High-performance preconditioning*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1174–1185.

ETNA
Kent State University
etna@mcs.kent.edu

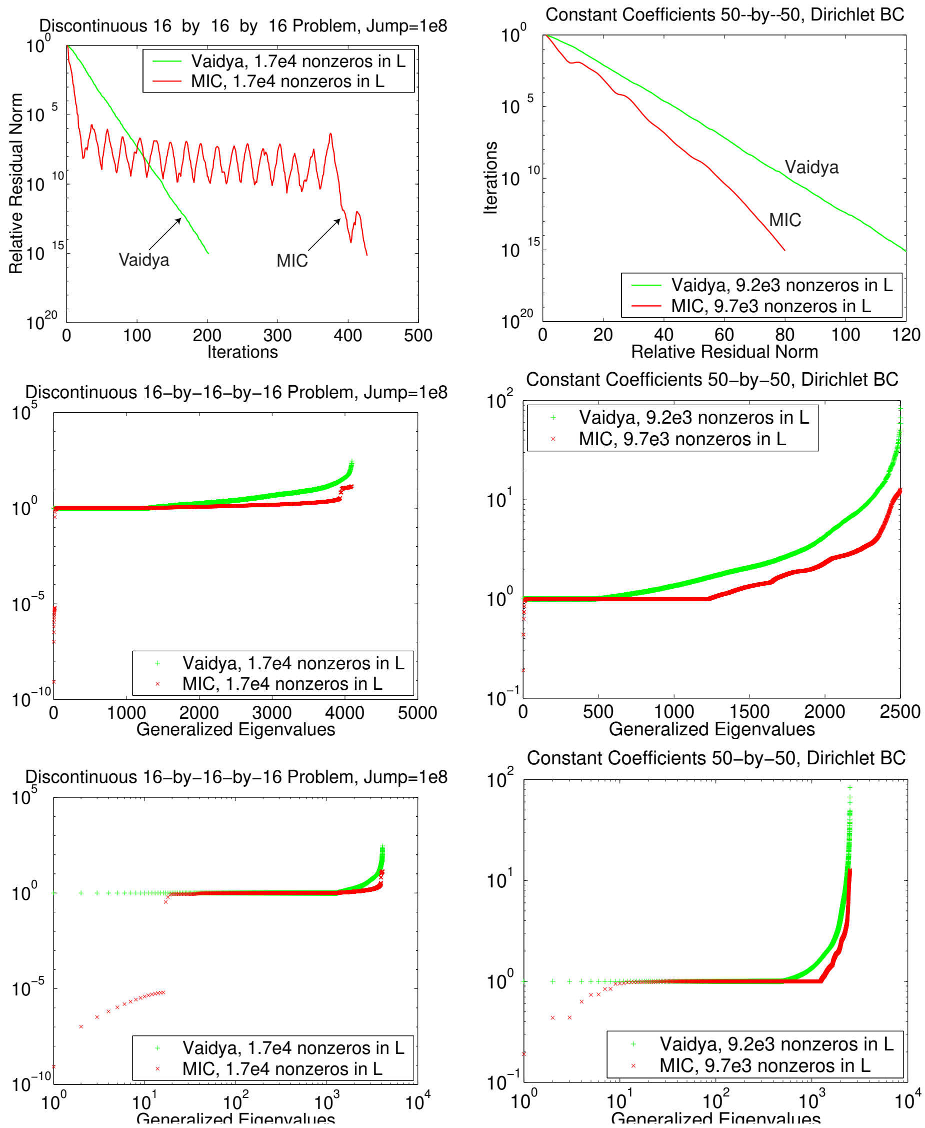


FIG. 3.3. *The convergence and generalized eigenvalues of Vaidya and relaxed-MIC preconditioners. The three graphs on the left show the behavior on a* 16*-by-*16*-by-*16 *problem with discontinuous coefficients and Neumann boundary conditions, and the three graphs on the right show the behavior on a* 50*-by-*50 *2D isotropic problems with Dirichlet boundary conditions. In both cases, the Vaidya and relaxed-MIC preconditioners have roughly the same number of nonzeros in L (indicated in the legend). In the 3D problem, Vaidya was ordered using GENMMD and MIC using the natural ordering, in the 2D problem, both were ordered using GENMMD. For each problem, the top graph plots the convergence (2-norm of the residual), the middle graph plots the eigenvalues on a linear-log scale, and the bottom graph plots the eigenvalues on a log-log scale, which allows the reader to observe the small eigenvalues. The generalized eigenvalues λ of $Ax = \lambda Mx$ in were computed by* MATLAB 6 *using* `eig(A,M,'chol')`*. The eigenvalues are presented from smallest (leftmost) to largest (rightmost).*

ETNA
Kent State University
etna@mcs.kent.edu

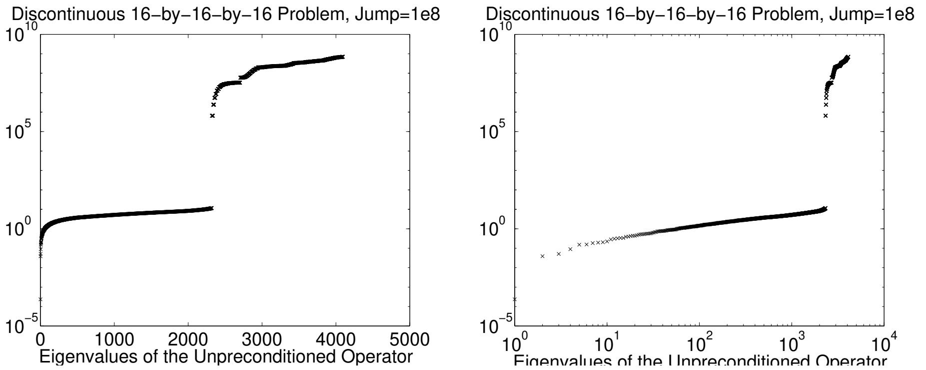


FIG. 3.4. *The eigenvalues of the* 16*-by-*16*-by-*16 *problem with discontinuous coefficients and Neumann boundary conditions, without preconditioning. The condition number is approximately* 3×10^{12}*. The graphs display the eigenvalues, sorted from smallest to largest, as computed by by* MATLAB *6 using* `eig(A)`*, on both a linear-log and a log-log scale.*

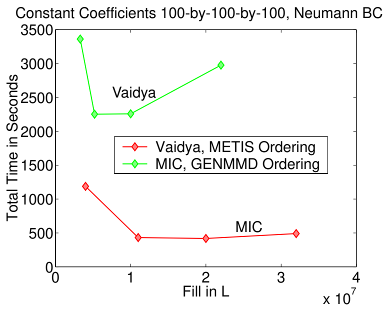


FIG. 3.5. *Total solution times as a function of the amount of fill for Vaidya and relaxed MIC on a 3D* 100*-by-*100*-by-*100 *isotropic problem with Neumann boundary conditions. In this experiment we reduced the norm of the residual by* 10^{15}*. We have obtained similar results in an experiment in which we reduced the residual by only* 10^{8}*. The same experiment is also described in Table 3.1.*

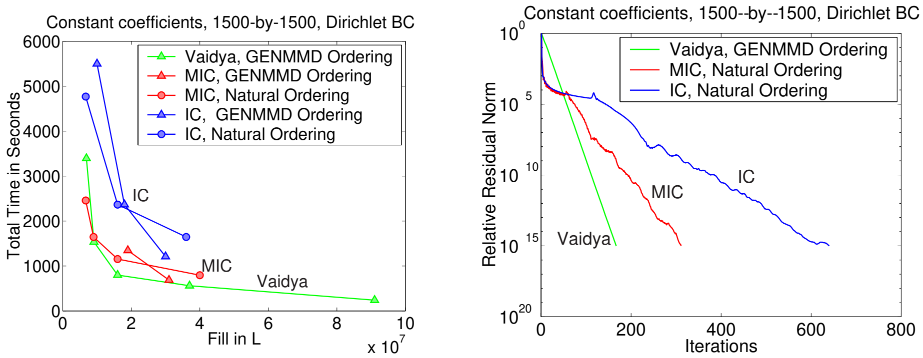


FIG. 3.6. *The performance of Vaidya, IC, and relaxed-MIC preconditioners on* 1500*-by-*1500 *2D isotropic problems with Dirichlet boundary conditions. The graph on the left shows the total solution time as a function of fill in the preconditioner. The rightmost data point on this graph represents a direct solver. The graph on the right shows the reduction in the norm of the residual as a function of the number of iterations, where both preconditioners have approximately* 1.6×10^{7} *nonzeros.*

ETNA
Kent State University
etna@mcs.kent.edu

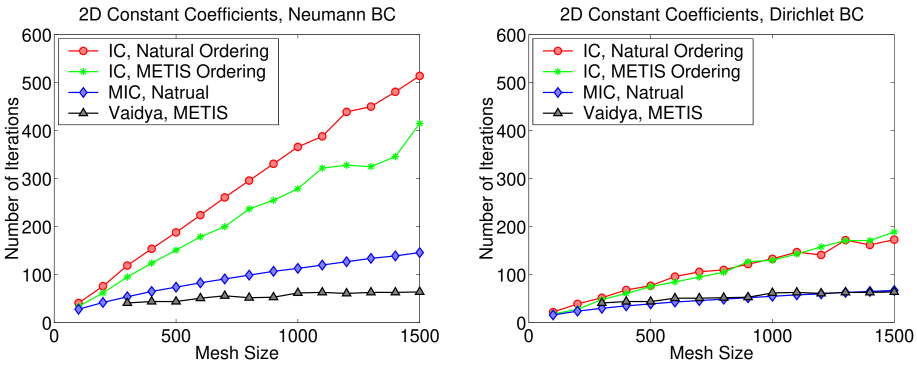


FIG. 3.7. *Convergence of Vaidya, IC, and (unrelaxed) MIC preconditioners on 2D isotropic problems with Neumann (left) and Dirichlet (right) boundary conditions. All the preconditioners have approximately* $10n$ *nonzeros in L. The graphs show the number of iterations it took to reduce the 2-norm of the residual by a factor of* 10^8 *as a function of the mesh size* $\sqrt{n}$ *(i.e., the matrices are* n*-by-*n*).*

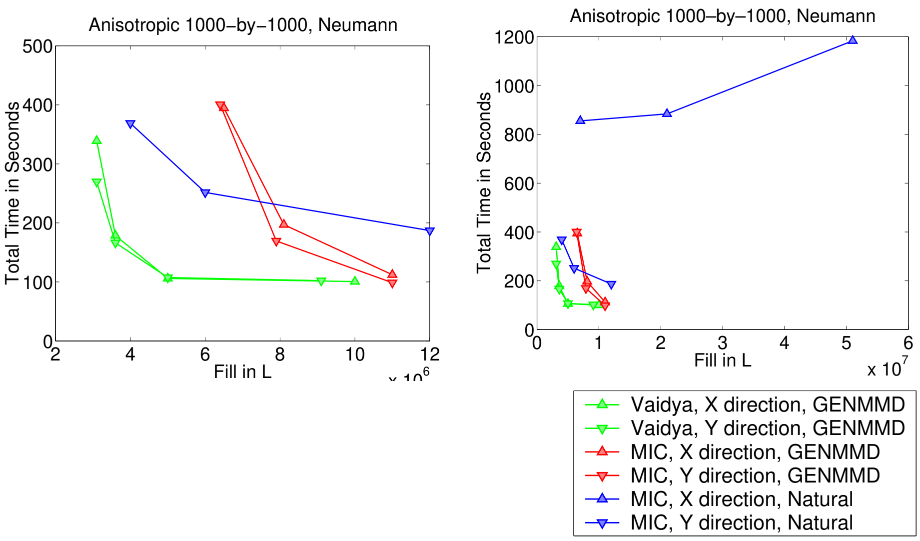


FIG. 3.8. *The performance of Vaidya and relaxed MIC on an anisotropic problem. The graph shows the total solution times for Vaidya with GENMMD ordering and for MIC with both GENMMD ordering and the natural ordering of the grid. The two graphs show the same data, and the legend on the bottom right applies to both, except that the graph on the left omits the naturally-ordered MIC with anisotropy in the x direction, in order to focus on the other cases.*

ETNA
Kent State University
etna@mcs.kent.edu

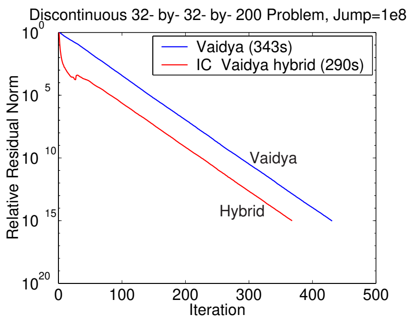


FIG. 4.1. *The convergence and total solution times of Vaidya's preconditioner with* 4.6×10^6 *nonzeros on a discontinuous 3D problem. The graph shows the convergence rate of Vaidya-preconditioned conjugate gradient and of a hybrid solver. The hybrid starts with 25 iterations of IC(0)-preconditioned conjugate gradient, and then uses the approximate solution as an initial guess in a Vaidya-preconditioned conjugate gradient. The graph plots the residual norm during both phases of this hybrid. The IC-Vaidya hybrid saves time in this case (from 343 seconds to 290), since the overall number of iterations is reduced, and since the costs of constructing and applying the IC(0) preconditioner are small.*