

THE INCOMPLETE FACTORIZATION MULTIGRAPH ALGORITHM

RANDOLPH E. BANK* AND R. KENT SMITH†

Abstract. We present a new family of multigraph algorithms, ILU-MG, based upon an incomplete sparse matrix factorization using a particular ordering and allowing a limited amount of fill-in. While much of the motivation for multigraph comes from multigrid ideas, ILU-MG is distinctly different from algebraic multilevel methods. The graph of the sparse matrix A is recursively coarsened by eliminating vertices using a graph model similar to Gaussian elimination. Incomplete factorizations are obtained by allowing only the fill-in generated by the vertex parents associated with each vertex. Multigraph is numerically compared with algebraic multigrid on some examples arising from discretizations of partial differential equations on unstructured grids.

Key words. hierarchical basis, algebraic multigrid, incomplete LU factorization.

AMS subject classifications. 65M55, 65N55

1. Introduction. In this paper, we present a new family of multigraph algorithms, ILU-MG, based upon an incomplete sparse matrix factorization using a carefully designed ordering and allowing a limited amount of fill-in. While in this paper we focus primarily on systems of linear equations arising from discretizations of partial differential equations, the method can be formally applied to general sparse matrices. For any particular problem or class of problems, it seems likely that specialized methods making use of the particular features of that problem will outperform any multigraph algorithm. However, the goal of the ILU-MG algorithm is to provide a general and robust iterative solver for many different systems of linear equations. While this goal may not yet be achieved in this first version, our hope and expectation is that the multigraph algorithm will eventually provide reasonably good rates of convergence for many classes of problems, while requiring only minimal input.

Algebraic approaches to multilevel methods have enjoyed a long history, beginning with the algebraic multigrid (AMG) methods of Brandt, McCormick, and Ruge [13, 14] Ruge and Stüben [26] and the black box multigrid method of Dendy [15]. More recent work can be found in [1, 3, 4, 12, 20, 19, 17], as well as many contributions in [2]. Our work grew out of the grid coarsening schemes developed in [10, 11] and the corresponding hierarchical basis iterations, HBMG. While much of our motivation comes from these multigrid ideas, ILU-MG is fundamentally an incomplete sparse matrix factorization.

The multigraph method resembles the approach of classical sparse Gaussian elimination. The graph of the stiffness matrix A is recursively coarsened by first eliminating a node with all its adjacent edges and then adding only a partial set of fill-in edges corresponding to the vertex parents of that node. Thus, there is no concept of levels, and more important, no coarse grid on which the problem must be solved exactly. However, the generality of this approach leaves open the possibility of introducing both levels and coarse graphs. We plan to study these alternatives as possible ways to improve the convergence behavior of the basic method. See [9] for some preliminary results in this direction. The size of each subsequent graph is controlled by monitoring the amount of numerical fill-in produced. Since each node is eliminated

* Department of Mathematics, University of California at San Diego, La Jolla, CA 92093. The work of this author was supported by the National Science Foundation under contract DMS-9706090.

† Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974.

based upon graph considerations, no attempt is made to preserve the integrity of the grid. Indeed, this information is not even provided.

The rest of this paper is organized as follows. In section 2, we provide a graph theoretic interpretation of the construction of hierarchical bases and its relation to sparse incomplete factorizations. The connection between multigrid and multigraph is described in terms of linear algebra in section 3. In section 4, the implementation of our method is discussed in some detail. In particular, the ordering strategy and the incomplete factorization procedure are described. Finally, in section 5, we compare multigraph with AMG on some examples arising from discretizations of partial differential equations on unstructured grids.

2. Graph theoretical aspects. In this section we discuss the relation between the construction of a hierarchical basis and sparse incomplete LU (ILU) factorization, within the context of graph theory. We first consider standard Gaussian elimination and classical ILU factorization from a graph theoretical point of view, and then develop a graph elimination model for hierarchical basis methods on sequences of nested meshes. These models can be interpreted as special ILU decompositions which generalize to the case of general graphs.

We begin with a few standard definitions; the interested reader is referred to Rose [25] or George and Liu [18] for a more complete introduction. Corresponding to a sparse $n \times n$ matrix A with symmetric sparsity pattern (i.e., $A_{ij} \neq 0$ if and only if $A_{ji} \neq 0$), let $G(V, E)$ be the graph that consists of a set of n ordered vertices $v_i \in V$, $1 \leq i \leq n$, and a set of edges E such that the edge $e_{ij} \in E$ (connecting vertices v_i and v_j) if and only if $A_{ij} \neq 0$, $i \neq j$. The edges in the graph G correspond to the nonzero off-diagonal entries of A . If A is the stiffness matrix for the space of continuous piecewise linear polynomials represented in the standard nodal basis, the graph G is just the underlying triangulation of the domain (with minor modifications due to Dirichlet boundary conditions). For vertex v_i the set of adjacent vertices $adj(v_i)$ is defined by

$$adj(v_i) = \{v_j \in V | e_{ij} \in E\}.$$

The degree of a vertex $deg(v_i)$ is just the size of the set $adj(v_i)$. A clique $C \subseteq V$ is a set of vertices which are all pairwise connected; that is, $v_i, v_j \in C, i \neq j \Rightarrow e_{ij} \in E$. With a proper ordering of the vertices, a clique corresponds to a dense submatrix of A . In graph theoretic terms, a single step of Gaussian elimination transforms $G(V, E)$ to a new graph $G'(V', E')$ as follows:

1. Eliminate vertex v_i and all its incident edges from G . Set $V' = V - \{v_i\}$. Denote the resulting set of edges $E_1 \subseteq E$.
2. Create a set F of fill-in edges as follows: for each distinct pair $v_j, v_k \in adj(v_i)$ in G , add the edge e_{jk} to F if not already present in E_1 . Set $E' = E_1 \cup F$.

Since the values of matrix entries are not involved, this model cannot take into account the occurrence of so-called *accidental zeros*. The graph elimination process is illustrated in Figure 1. Note that the set $adj(v)$ in G becomes a clique in G' . Within this framework, the classical ILU factorization is one in which *no* fill-in edges are allowed, i.e., $F \equiv \emptyset$. This forces the matrix A' corresponding to the new graph G' to have the same sparsity structure as the corresponding submatrix of A . The graph G' would then correspond to the center picture in Figure 1.

The concept of *vertex parents* is first introduced to allow HBMG to be interpreted as a generalized ILU procedure. We will begin with the case of two nested meshes where the fine mesh is a uniform refinement of a coarse mesh, generated by pairwise

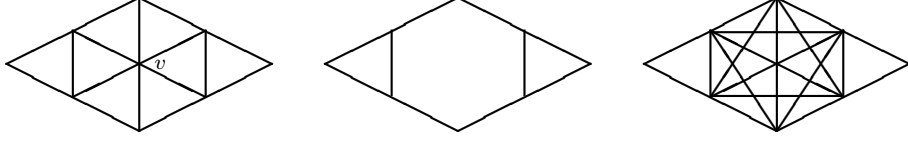


FIG. 1. A fine grid vertex v is eliminated by classical Gaussian elimination. The original mesh is shown on the left. We remove v and its incident edges (middle), and add fill-in edges (right).

connecting the midpoints of the coarse grid edges in the usual way [7, 27, 21]. Here we can make the direct sum decomposition $V = V_c \oplus V_f$, where V_c is the set of coarse grid vertices and V_f is the set of fine grid vertices (those not in V_c). For each vertex $v_i \in V_f$, there is a unique pair of vertex parents $v_j, v_k \in V_c$ such that v_i is the midpoint of the edge connecting v_j and v_k ($v_i = (v_j + v_k)/2$).

We now view HBMG as an *ILU* algorithm in which only selected fill-in edges are allowed. In this algorithm, the vertices in the set V_f are sequentially eliminated as follows:

1. Eliminate vertex $v_i \in V_f$ and all its incident edges from G . Set $V' = V - \{v_i\}$. Denote the resulting set of edges $E_1 \subseteq E$.
2. Let $v_k, v_j \in \text{adj}(v_i)$ denote the parents of v_i . Create a set F of fill-in edges of the form e_{jm} , $m \neq j$ or e_{km} , $m \neq k$, $v_m \in \text{adj}(v_i)$ for edges not already present in E_1 . Set $E' = E_1 \cup F$.

In other words, the classical HBMG algorithm adds the subset of fill-in edges from Gaussian elimination in which one of the vertices is a vertex parent.

An even more simple possibility is to use just *one* vertex parent as an elimination strategy. Although this does not correspond to the classical case of HBMG, it has been studied in a different context as a partitioning scheme for general graphs [23, 24]. The elimination algorithm is similar to the case of two parents:

1. Eliminate vertex $v_i \in V_f$ and all its incident edges from G . Set $V' = V - \{v_i\}$. Denote the resulting set of edges $E_1 \subseteq E$.
2. Let $v_j \in \text{adj}(v_i)$ denote the parent of v_i . Create a set F of fill-in edges of the form e_{jm} , $m \neq j$, $v_m \in \text{adj}(v_i)$ for edges not already present in E_1 . Set $E' = E_1 \cup F$.

However, in this case generally fewer fill-in edges are added. When the initial graph G is a finite element triangulation (or a tetrahedral mesh in three space dimensions), the graph G' remains a finite element triangulation. This property can be maintained at *all* steps of the elimination process through a careful selection of parents. Both one and two vertex parent eliminations are illustrated in Figure 2.

Another straightforward extension allows for the possibility of more than two vertex parents, either a fixed or variable number. The limiting case, allowing *all* vertices in $\text{adj}(v)$ to be parents of v , will result in classical Gaussian elimination. These extensions can be interpreted, in the multigrid framework, as various multipoint interpolation schemes. In this work, we consider only the one- and two-vertex parent cases.

Let the triangulation \mathcal{T}_f be the graph for the original stiffness matrix A represented in the standard nodal basis. For either one or two parents, after all the

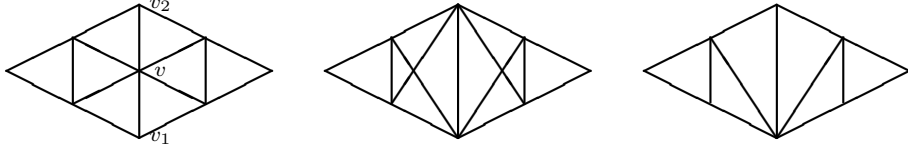


FIG. 2. A fine grid vertex v is eliminated by HBMG ILU elimination. The original mesh is shown on the left. The fill-in pattern using two parents (v_1 and v_2) is shown in the middle, while the fill-in pattern using the single parent v_1 is shown on the right.

vertices in V_f are eliminated, the resulting graph is just the coarse grid triangulation \mathcal{T}_c . However, the numerical values of the matrix elements are generally different for the two cases. For the special case of a sequence of uniformly refined meshes, the total number of edges in the filled in graph can be estimated. This will serve as a guide to the amount of memory necessary to store the incomplete LU factorization using typical sparse matrix storage schemes, e.g., [16, 18, 8]. The elimination process is illustrated for the case of two parents in Figure 3.

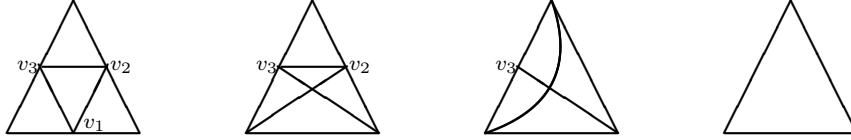


FIG. 3. HBMG/ILU in the classical case, using two parents. The fine grid vertices are eliminated in the order v_1 , v_2 , v_3 . A total of two fill-in edges are added in the interior of the coarse grid triangle during the elimination.

The original graph has approximately $3n$ edges and $2n$ triangles. Each quartet of four triangles generates two fill-in edges which are not part of the coarse grid triangulation. Thus the total number of edges *not* in the coarse grid is approximately $3n + 2(2n/4) = 4n$. If we repeat recursively for the coarser grids, the geometric sum

$$4n \left\{ 1 + \frac{1}{4} + \frac{1}{16} + \dots \right\} \approx 4n \frac{4}{3} = \frac{16n}{3}$$

is generated. Thus there are approximately $16n/3$ edges in the filled in graph. In the case of classical HBMG, the method is implemented as a *block* iteration, with the fill-in in the off-diagonal blocks not explicitly stored. The number of edges actually stored in the sparse data structure is approximately $3n(4/3) = 4n$, as shown in [7]. In our present study, we consider only point iterations with explicit storage of all edges. The elimination process for the case of one parent is shown in Figure 4.

The main difference in the case of one parent is that now only one fill-in edge is associated with each quartet of triangles on the fine grid. Thus we have a total of approximately $3n + 2n/4 = 7n/2$ edges not in the coarse grid and expect approximately

$$\frac{7n}{2} \left\{ 1 + \frac{1}{4} + \frac{1}{16} + \dots \right\} \approx \frac{7n}{2} \frac{4}{3} = \frac{14n}{3}$$

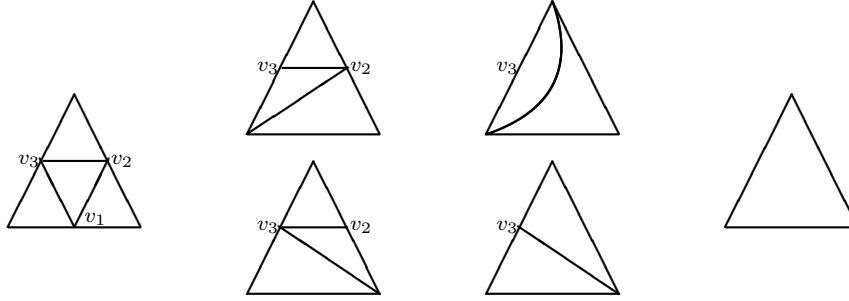


FIG. 4. *HBMG ILU using one parent. The fine grid vertices are eliminated in the order v_1, v_2, v_3 . There are eight generic scenarios for the process, but all result in one fill-in edge in the interior of the coarse grid triangle.*

total edges, somewhat less than the case of two parents. This is also less than the bound of $6n$ obtained for general matching strategies on two-dimensional triangulations [23]. In fact, matching strategies when applied to general graphs produce fill-in bounded by $O(|E_0| \ln n)$ where E_0 is the number of edges in the original graph. We know of no similar bound for the case of two parents, and have empirically observed very rapid growth in fill-in for some matching strategies which made little effort to control the number of fill-in edges.

The multigraph algorithm generalizes the concept of vertex parents to arbitrary graphs. The main problem is to determine reasonable vertex parents for each vertex to be eliminated. Once this is done, the elimination/unrefinement/coarsening is performed on the graph exactly as in the case of nested meshes.

3. The multigraph algorithm. The salient features of the multigraph algorithm and its connection to HBMG can be illustrated by considering the following simple example. Let A be an $n \times n$ sparse matrix arising from the discretization of a partial differential equation, assembled using the standard nodal basis. We partition A as

$$(1) \quad A = \begin{pmatrix} d & r^t \\ c & B \end{pmatrix},$$

where $d \neq 0$ is the diagonal matrix element for vertex v_1 , r and c are vectors of order $n - 1$, and B is an $(n - 1) \times (n - 1)$ matrix. The hierarchical basis multigrid method is based on a change of basis, from the nodal basis to a hierarchical basis. In the present context, the first step of this process involves forming the matrix

$$(2) \quad \begin{aligned} L_1 A U_1 &= \begin{pmatrix} 1 & 0 \\ \ell & I \end{pmatrix} \begin{pmatrix} d & r^t \\ c & B \end{pmatrix} \begin{pmatrix} 1 & u^t \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} d & r^t + d u^t \\ c + d \ell & B + \ell r^t + \ell d u^t + c u^t \end{pmatrix}. \end{aligned}$$

The vectors ℓ and u are sparse with nonzeros determined by the vertex parents of vertex v_1 .

For classical HBMG, v_1 is a vertex associated with the refinement of an edge with

endpoints $\{v_i, v_j\}$. In this case,

$$\ell = u = \frac{e_i + e_j}{2}$$

where e_j is the usual unit vector (j th column of the identity matrix I_{n-1}). Here the restriction and interpolation operators are chosen geometrically, reflecting the fact that $v_1 = (v_i + v_j)/2$.

One possible generalization is to choose the vectors ℓ and u based on an incomplete factorization of the matrix A . In particular, let v_i and v_j be the parents of vertex v_1 and

$$\ell = \ell_i e_i + \ell_j e_j \quad \text{and} \quad u = u_i e_i + u_j e_j.$$

The scalars ℓ_i, ℓ_j and u_i, u_j are simply the multipliers of an incomplete LU factorization. Note that the above equations are well defined, independent of the geometry of the mesh, and they *do not* require that v_1 results from the refinement of an edge e_{ij} . In a similar manner, for the case of one vertex parent we have

$$\ell = \ell_i e_i \quad \text{and} \quad u = u_i e_i.$$

The sparsity patterns of the vectors c and $c + d\ell$ are the same, since the nonzeros in ℓ are a subset of the nonzeros in c .¹ However, the sparsity patterns of B and $B + \ell du^t + lr^t + cu^t$ generally are not the same. The matrix $\ell du^t + lr^t + cu^t$ typically creates some fill-in in the rows and columns corresponding to the vertex parents. These are precisely the fill-in edges illustrated in Figure 2. Also note that

$$B + lr^t + \ell du^t + cu^t = B - cd^{-1}r^t + (\ell + cd^{-1})d(u + rd^{-1})^t.$$

From this identity, we can see that this elimination step can also be viewed as forming a rank one perturbation of the exact Schur complement $B - cd^{-1}r^t$.

In its standard formulation, the next step of this hierarchical decomposition is a transformation of the same form applied to the reduced matrix $B + lr^t + \ell du^t + cu^t$ (not $L_1 A U_1$). The actual change to the hierarchical basis is defined *implicitly* through this recursion. The final hierarchical bases stiffness matrix A' is far less sparse, but generally better conditioned than A , so that standard iterative methods can be effectively applied to linear systems involving A' . In the case of classical HBMG, the iterative method is just a standard block symmetric Gauss–Seidel iteration, with the blocks defined in terms of refinement levels in the mesh. See [6, 7] for details.

The multigraph method, ILU-MG, replaces this Gauss–Seidel iteration with an incomplete factorization. In particular we set

$$(3) \quad A = LDU + E,$$

where L is unit lower triangular, D is diagonal, U is unit upper triangular, and E is the so-called *error matrix*. The sparsity pattern of $L + D + U$ is defined in terms of the symbolic elimination algorithm described in section 2. This is precisely the sparsity pattern recursively generated by the hierarchical transformations defined above. By using a standard ILU factorization, we avoid the recursion of HBMG. At the same time, we hope that by allowing this additional fill-in, ILU-MG will inherit the desirable properties of HBMG as a preconditioner.

¹ Here we are speaking in generic terms and in particular are not taking into account the possible occurrence of so-called *accidental zeros*.

4. Implementation. Our multigraph algorithm is divided into four distinct phases, analogous to classical sparse Gaussian elimination algorithms.

1. Ordering: A permutation matrix P is computed to reorder the matrix, PAP^t . In addition, vertex parents for each eliminated vertex are defined and determine the fill-in pattern for the second phase.
2. Symbolic factorization: The (incomplete) fill-in is computed using the graph of PAP^t and the vertex parents. The output is a static data structure for the incompletely factored sparse matrix.
3. Numerical factorization: The numerical values of the L , D , and U factors are computed using a *MILU* factorization.
4. Solution: The solution of $Ax = b$ is computed using a conjugate or biconjugate gradient algorithm, preconditioned by the incomplete factorization.

This section is divided into two parts. First, we describe the ordering strategy used to compute P and the vertex parents for each vertex. Next, our *MILU* factorization is discussed. We do not believe that our present algorithms for these two phases are optimal in any sense. However, they are the best ones we have found so far and their performance seems to justify further work in this area.

4.1. Ordering. In the case of classical sparse Gaussian elimination, ordering consists of finding a permutation matrix P such that the reordered matrix PAP^t has some desired property in terms of the ensuing factorization. Normally, the permutation matrix P is constructed based solely on the graph of the matrix (e.g., a minimum degree ordering [16, 25, 18]) and not on the values of the matrix elements. In the multigraph algorithm, both the graph and the numerical values of the matrix A are used to construct both the ordering and the vertex parents.

To simplify notation, we will describe only how the first vertex is ordered and its parents are selected. The remaining vertices are ordered by the same algorithm applied inductively. Let

$$\rho_i = |a_{ii}| + \sum_{j \neq i}^n |a_{ij}| + |a_{ji}|.$$

For $\gamma > 0$, the quality r_γ^{ij} is given by

$$(4) \quad r_\gamma^{ij} = \frac{|a_{ii}| + |a_{ij}|}{\rho_i(f^{ij} + \gamma)},$$

where f^{ij} is the number of fill-in edges which must be added if $v_j \in \text{adj}(v_i)$ is chosen as the only vertex parent of v_i . For the case of one vertex parent, the quality function $q_1(v_i)$ is defined by

$$(5) \quad q_1(v_i) = \max_{v_j \in \text{adj}(v_i)} r_\gamma^{ij}.$$

The *tentative* vertex parent is the vertex $v_j \in \text{adj}(v_i)$ such that $r_\gamma^{ij} = q_1(v_i)$. Ties are broken arbitrarily. The quality function $q_1(v_i)$ represents a compromise between choosing a parent v_j which is as strongly connected to v_i as possible (measured in terms of the size of the off-diagonal matrix elements), and choosing v_j to cause as little fill-in as possible in terms of the factorization.

The size of the parameter γ can be used to indirectly control the number of fill-in edges resulting from the ordering. Smaller values for γ result in less fill-in. Experimentally, we determined $\gamma = 10$ to be a good choice.

The quality function $q_2(v_i)$ for the case of two vertex parents is developed in a similar fashion. Suppose that $v_j, v_k \in \text{adj}(v_i)$ ($k \neq j$). Then the qualities s_γ^{ijk} are given by

$$(6) \quad s_\gamma^{ijk} = \frac{|a_{ii}| + |a_{ij}| + |a_{ki}|}{\rho_i(f^{ij} + f^{ik} + \gamma)}.$$

Let g^{ijk} denote the number of fill-in edges required if $\{v_j, v_k\}$ are chosen as parents; then $g^{ijk} \leq f^{ij} + f^{ik} \leq g^{ijk} + 1$. Let

$$\hat{q}_0(v_i) = \frac{|a_{ii}|}{\rho_i \gamma}, \quad \hat{q}_1(v_i) = \max_{v_j \in \text{adj}(v_i)} r_\gamma^{ij}, \quad \hat{q}_2(v_i) = \max_{v_j, v_k \in \text{adj}(v_i)} s_\gamma^{ijk},$$

and set

$$(7) \quad q_2(v_i) = \max\{\hat{q}_0(v_i), \hat{q}_1(v_i), \hat{q}_2(v_i)\}.$$

Our two parent algorithm actually offers the possibility to each vertex of having zero, one, or two parents. The choice of parents is based on maximizing the function $q_2(v_i)$. As in the single parent case, the quality function $q_2(v_i)$ seeks a compromise between choosing strongly connected parents, and choosing parents which allow low fill-in; experimentally, we determined $\gamma = 50$ for the two parent algorithm.

We now describe our algorithm for ordering vertices and computing vertex parents. We begin by computing the quality $q_p(v_i)$ for each vertex in the mesh, using (5) for the case $p = 1$ (one vertex parent algorithm) or (7) for the case $p = 2$ (two vertex parent algorithm). Along with the quality function, *tentative* parents are assigned to each vertex. In cases where no parents can be assigned, $q_p(v_i) = 0$. The vertices are then placed in a heap with the vertex of highest quality at the root. This vertex, say v_i , is ordered first, and its tentative parents become its *actual* parents. We then update the graph and compute the reduced matrix. The quality function and heap position for vertices $v_j \in \text{adj}(v_i)$ are updated, as vertices in this set are the only ones potentially affected by the elimination of v_i . This process continues inductively until all the vertices are ordered (the usual case), or all remaining vertices have $q_p(v_k) = 0$ (in which case an arbitrary order is assigned to the remaining vertices). We summarize this algorithm below:

1. Initialize by computing $q_p(v_i)$ and tentative parents for v_i , $1 \leq i \leq n$. Place all vertices in a heap according to $q_p(v_i)$.
2. While the heap is not empty and the root vertex has $q_p(v_i) > 0$, do i–iii below.
 - i. Order the vertex v_i at the root of the heap; update the heap. The tentative parents of v_i become the actual parents.
 - ii. Eliminate v_i from the current graph; add fill-in edges as required. Update the partially factored matrix using the *MILU* decomposition.
 - iii. For $v_j \in \text{adj}(v_i)$, update $q_p(v_j)$, and update the position of v_j in the heap.

We note that step ii above essentially requires an incomplete factorization of the matrix A to occur concurrently with the ordering, since the quality function q_p is updated based on the current state of the factorization. This makes the ordering algorithm rather expensive, often as expensive as or even more expensive than the actual solution. This is partly because this factorization must use a dynamic data

structure, rather than the static sparse matrix data structures which we employ elsewhere. On the other hand, as in general sparse matrix calculations, in many cases ordering can be done once and then used for several factorizations and solutions.

There are several interesting variations of our ordering algorithm which merit some discussion. Both are related to step iii above and provide a means of partitioning the matrix in order to formulate block iterative methods. First, in step iii, we can (artificially) set $q_p(v_j) = 0$ if $v_j \in \text{adj}(v_i)$ is a parent of v_i . This forces v_j (and all other vertices chosen as actual parents) to reside near the bottom of the heap. When the heap contains only vertices with $q_p = 0$, the remaining vertices are called *coarse graph* vertices and those eliminated are *fine graph* vertices. This effectively provides a two level blocking in a fashion quite analogous to the classical two level HBMG algorithm. If we (correctly) reinitialize $q_p(v_i)$ and compute tentative parents for all vertices remaining in the heap, and then restart the elimination process, we are led to a natural multilevel blocking, which could form the basis of a block incomplete factorization algorithm.

Second, in step iii we can artificially set $q_p(v_j) = 0$ for *all* $v_j \in \text{adj}(v_i)$. Then the fine graph (eliminated) vertices will form an independent set, in that the diagonal block of both the original and factored matrices corresponding to this set of vertices will be diagonal. Reinitializing and restarting the elimination process as above would result in some multicolor-like ordering, which might have some interesting applications for vector or parallel processing.

For either of these alternatives, using an enhanced quality function that includes additional information about the blocking strategy (e.g., bias q_p to favor producing the largest number of vertices in the fine graph set within the other constraints) seems appropriate.

4.2. Numerical factorization. Our implementation of the *MILU* factorization is defined as follows. Let

$$(8) \quad \sigma_i = \max \left\{ 0, - \sum_{j=1}^n (a_{ij} + a_{ji}) \right\}$$

and set $\Sigma = \text{diag}(\sigma_i)$. We actually compute an incomplete factorization of the matrix $A + \Sigma$. Making such an a priori shift is one simple way to ensure the existence and stability of our factorization. To describe the first step of our factorization procedure, let

$$(9) \quad A + \Sigma = \begin{pmatrix} d & r^t \\ c & B \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ c/d & I \end{pmatrix} \begin{pmatrix} d & 0 \\ 0 & B - cr^t/d \end{pmatrix} \begin{pmatrix} 1 & r^t/d \\ 0 & I \end{pmatrix}.$$

The sparsity pattern of matrix $-cr^t/d$ generally will not coincide with the sparsity pattern we choose to allow. Thus we set $-cr^t/d = S_1 + E_1$ where S_1 has the required sparsity pattern and E_1 is the *error matrix* for the first step. If we (inductively) continue the factorization as $B + S_1 = \hat{B} = \hat{L}\hat{D}\hat{U} + \hat{E}$, we have

$$(10) \quad \begin{aligned} A + \Sigma &= \begin{pmatrix} 1 & 0 \\ c/d & I \end{pmatrix} \begin{pmatrix} d & 0 \\ 0 & \hat{B} + E_1 \end{pmatrix} \begin{pmatrix} 1 & r^t/d \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ c/d & \hat{L} \end{pmatrix} \begin{pmatrix} d & 0 \\ 0 & \hat{D} \end{pmatrix} \begin{pmatrix} 1 & r^t/d \\ 0 & \hat{U} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \hat{E} + E_1 \end{pmatrix} \\ &= LDU + E'. \end{aligned}$$

Then we have

$$(11) \quad A = LDU + E' - \Sigma = LDU + E.$$

The matrix $-cr^t/d$ is decomposed as $S_1 + E_1$ by a procedure similar to classical *MILU* [22]. Suppose that $p > q$ and the element $c_p r_q/d$ is not allowed in the fill-in pattern. Consider the matrix F^{pq} which is zero except for the four elements

$$F_{pq}^{pq} = -c_p r_q/d, \quad F_{qp}^{pq} = -c_q r_p/d, \quad F_{pp}^{pq} = F_{qq}^{pq} = (c_q r_p + c_p r_q)/(2d).$$

We then set

$$E_1 = \sum F^{pq}, \quad S_1 = -cr^t/d - E_1$$

where the sum is taken over all (p, q) pairs falling outside the allowed fill-in pattern. This is a typical *MILU* approach, although the averaging of diagonal entries appearing in F^{pq} for the nonsymmetric case is a bit unusual.

5. Numerical results. In this section we present a few numerical results for a first version of our multigraph algorithm. We compare both the one parent and two parent versions of multigraph with the well-known algebraic multigrid code *AMG* of Ruge, Stüben, and Hempel. A suite of six text problems were constructed using the *PLTMG* package, version 7.8 [5].

For each problem, three nonuniform adaptive grids with $N = 5000, 20000, 80000$ were generated. For each test case, both the sparse matrix and the right hand side were saved in a file to serve as input for the iterative solvers.² The specific definition of each test problem is described below.

Problem Superior. This problem is a simple Poisson equation

$$-\Delta u = 1$$

with homogeneous Dirichlet boundary conditions on a domain in the shape of Lake Superior. This is the classical problem on a fairly complicated domain. The solution, shown in Figure 5, is generally very smooth but has some boundary singularities.

Problem Hole. This problem features discontinuous, anisotropic coefficients. The domain consists of three subregions. On the inner region, the problem is

$$-\epsilon \Delta u = 0$$

with $\epsilon = 10^{-2}$. In the middle region, the equation is

$$-\Delta u = 1,$$

and in the outer region, the equation is

$$-u_{xx} - \epsilon u_{yy} = 1.$$

Homogeneous Dirichlet boundary conditions are imposed on the inner (hole) boundary, homogeneous Neumann conditions on the outer boundary, and the natural continuity conditions on the internal interfaces. While the solution, shown in Figure 5, is also relatively smooth, singularities exist at the internal interfaces.

² These files are available upon request.

Problem Texas. This is an indefinite Helmholtz equation

$$-\Delta u - 2u = 1$$

posed in a region shaped like the state of Texas. Homogeneous Dirichlet boundary conditions are imposed. The length scales of this domain are roughly 16×16 , so this problem is fairly indefinite, as illustrated in Figure 6.

Problem UCSD. This is a simple constant coefficient convection-diffusion equation

$$-\nabla \cdot (\nabla u + \beta u) = 1,$$

$\beta = (0, 10^5)^T$ posed on a domain in the shape of the UCSD logo. Homogeneous Dirichlet boundary conditions are imposed. As seen in Figure 6, boundary layers are formed at the bottom of the region and the top of the obstacles.

Problems Jcn 0 and Jcn 180. The next two problems are solutions of the current continuity equation taken from semiconductor device modeling. This equation is a convection-diffusion equation of the form

$$-\nabla \cdot (\nabla u + \beta u) = 0.$$

The domain has seven subregions; $\beta = 0$ in the upper left and large lower region. In the narrow curved band, $|\beta| \approx 10^4$, and is directed radially. Dirichlet boundary conditions $u = 10^{-5}$ and $u = 10^{10}$ are imposed along the bottom boundary and along a short segment on the upper left boundary, respectively. Homogeneous Neumann boundary conditions are specified elsewhere. The solutions, see Figure 7, vary exponentially across the domain which is typical of semiconductor problems.

In the first problem, Jcn 0, the convective term is chosen so the device is *forward biased*. In this case, a sharp internal layer develops along the top interface boundary. In the second problem, Jcn 180, the sign of the convective term is reversed, resulting in two sharp internal layers along both interface boundaries.

All problems were run on a SGI R10000 Octane with 256 Mb of memory and compiled with the Fortran f77 -O -64 options. Each run consisted of two phases. The setup phase consisted of performing several initialization steps. For multigraph algorithms, this included ordering, symbolic factorization, and numerical factorization. Of these three steps, the sparse ordering is by far the most dominant. The initialization step for *AMG* consisted of determining the multigrid levels and constructing the interpolation operators as well as the coarse grid matrices. In the second phase, each problem was solved to a relative accuracy in the residual of 10^{-6} starting from an initial guess of zero.

The results of this comparison are shown in Table 1 and Table 2. Since there was little variation in the timings for the setup phase, these times (in seconds) are averaged over all problems with the same grid size.

TABLE 1
Average setup time vs. problem size.

N	AMG	One parent	Two parents
5 k	0.17	0.20	0.52
20 k	0.97	1.30	3.16
80 k	6.40	10.05	22.57

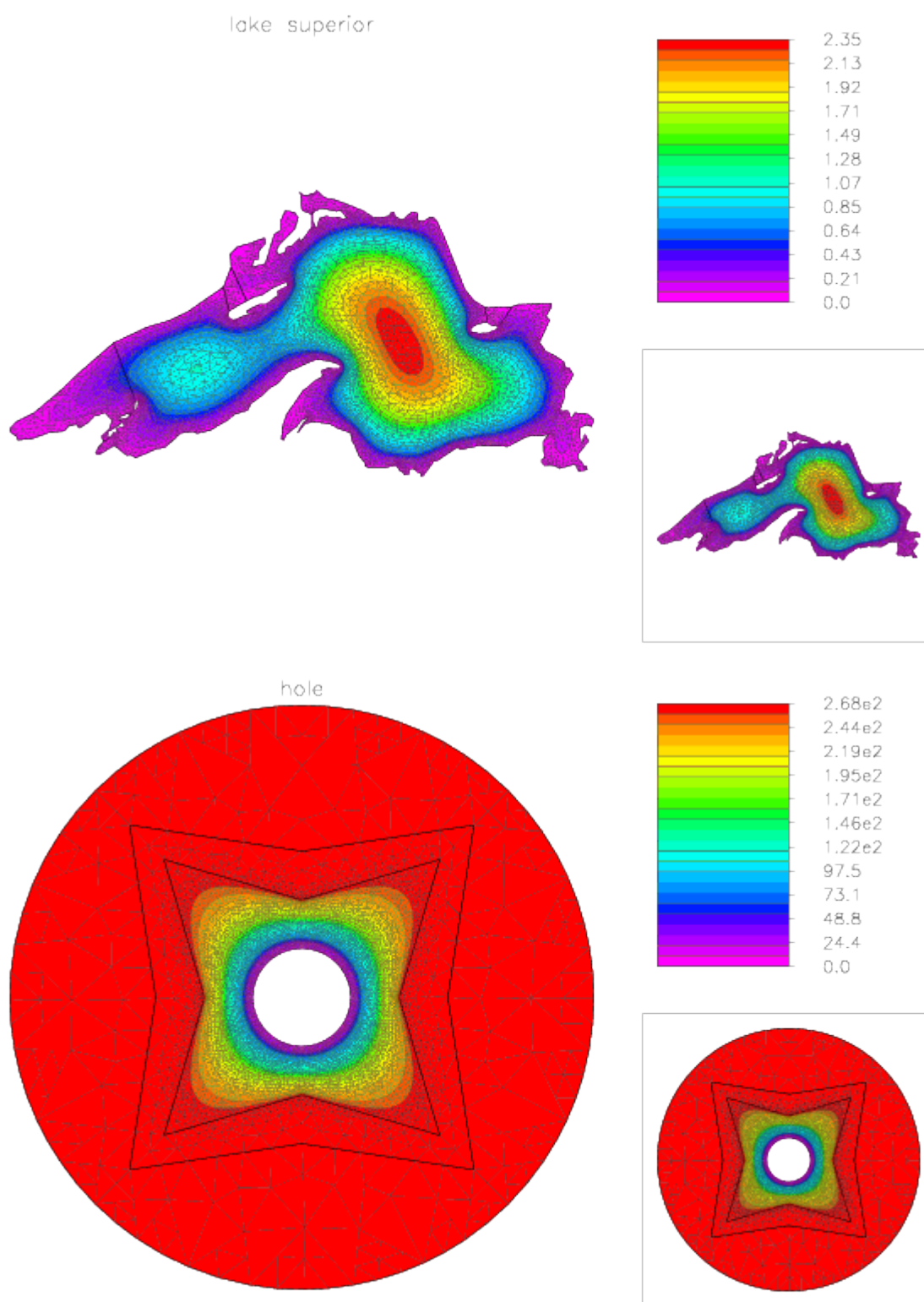


FIG. 5. *Lake Superior and Hole problems.*

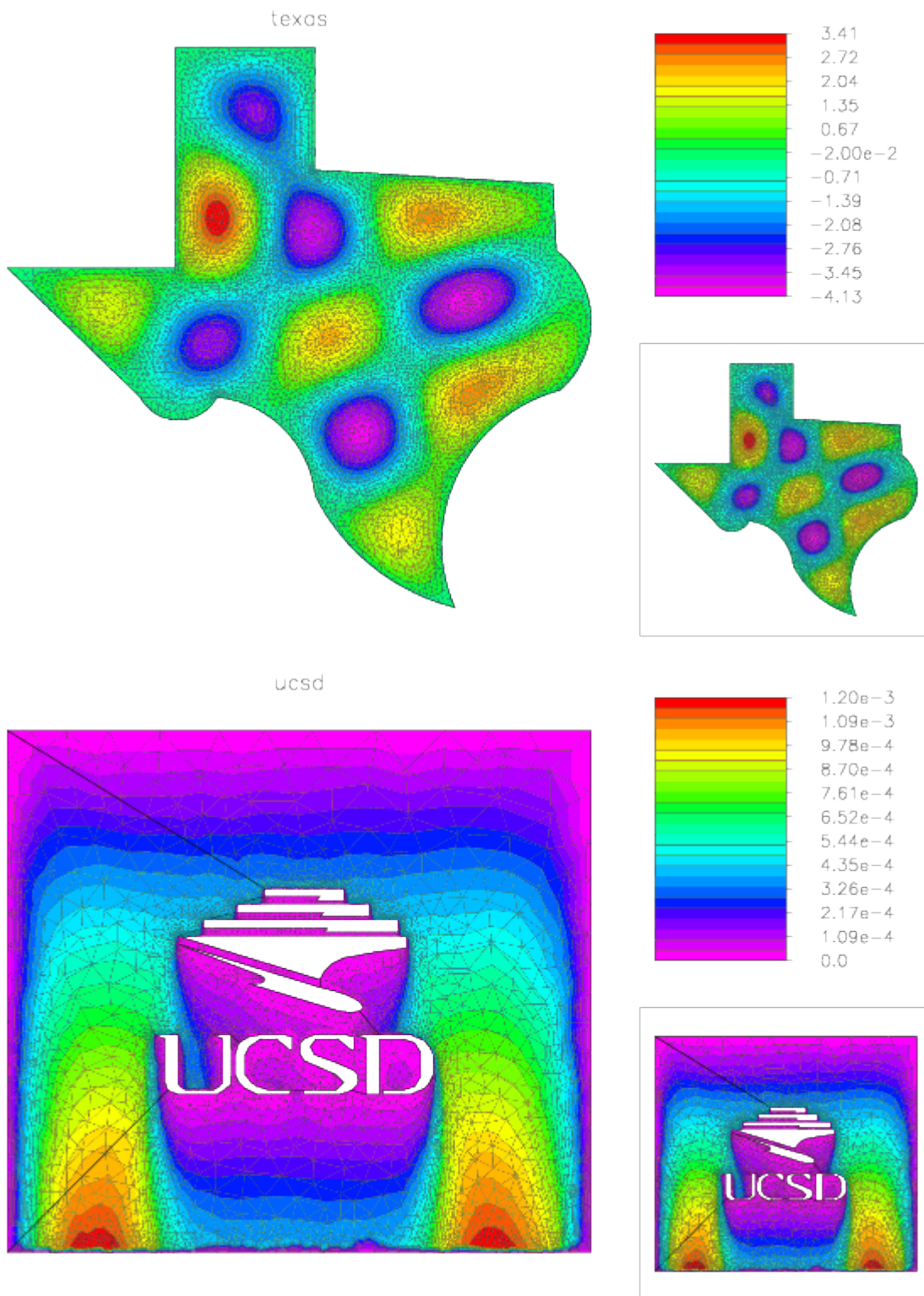


FIG. 6. *Indefinite and boundary layer problems.*

FIG. 7. *Semiconductor convection-diffusion problems.*

In Table 2, we present the number of *AMG* cycles and multigraph solves. The multigraph algorithm is used as a preconditioner for the composite step conjugate gradient (CSCG) procedure for symmetric problems of the composite step biconjugate gradient (CSBCG) procedure for nonsymmetric problems. We count solves rather than iterations, since composite steps cost about twice as much as single steps. Also, solves for nonsymmetric problems are twice as expensive as for symmetric problems, since both A and A^t must be preconditioned. The *digits* columns refer to

$$digits = -\log(\|r_k\|_{\ell^2}/\|r_0\|_{\ell^2}),$$

where r_k is the residual at the k th iteration (cycle). To compare these methods, we have chosen the CPU time, measured in seconds, to solve each problem.

TABLE 2
Performance comparison.

N	AMG			One parent			Two parents		
	Cycles	Digits	Time	Solves	Digits	Time	Solves	Digits	Time
Superior									
5 k	8	6.7	0.17	11	6.0	0.11	9	6.7	0.10
20 k	10	6.5	1.33	16	6.1	0.79	12	6.6	0.70
80 k	12	6.5	8.07	27	6.3	6.14	15	6.5	4.10
Hole									
5 k	49	6.0	1.24	35	6.1	0.38	14	6.0	0.17
20 k	48	6.0	6.76	53	6.2	2.74	19	6.1	1.15
80 k	32	6.0	22.10	128	6.0	28.90	26	6.3	7.29
Texas									
5 k	50	-34.5	1.65	85	6.1	0.83	74	6.3	0.81
20 k	50	-38.7	49.80	113	6.0	5.44	83	6.1	4.65
80 k	50	-52.0	113.00	166	6.1	37.60	91	6.1	24.30
UCSD									
5 k	5	7.2	0.12	8	6.7	0.14	7	6.3	0.14
20 k	5	6.4	0.71	10	6.3	0.93	9	6.3	0.85
80 k	5	6.5	3.67	12	6.3	5.04	11	6.1	5.05
Jcn 0									
5 k	50	-1.3	1.37	93	6.1	1.69	70	6.2	1.58
20 k	49	6.1	7.45	96	6.3	9.28	64	6.2	6.99
80 k	11	6.1	8.11	119	6.1	51.10	68	6.0	34.60
Jcn 180									
5 k	50	-0.3	1.37	10	6.0	0.20	7	6.5	0.18
20 k	50	-6.7	7.67	10	6.2	1.01	7	6.5	0.88
80 k	13	6.2	9.62	12	6.2	5.54	8	6.0	4.58

The timings for the one parent and the two parent versions exhibit a greater than linear growth as a function of problem size. For conventional test problems on

uniform 5-point grids, we observe an $O(N \log N)$ dependence, which is significantly better than most other *ILU* methods. In contrast, *AMG* is observed to be $O(N)$ for this class of problems. However, for all the problems we considered in this study, our algorithm is competitive with *AMG*.

Acknowledgments. The authors wish to thank Steve McCormick for helpful comments and John Ruge for his invaluable assistance in running the *AMG* code.

REFERENCES

- [1] O. AXELSSON AND M. NEYTCHIEVA, *The algebraic multilevel iteration methods - theory and applications*, in Proceedings of the Second International Colloquium in Numerical Analysis, Plovdiv, Bulgaria, 1993, pp. 13–23.
- [2] O. AXELSSON AND B. POLMAN, *AMLI'96: Proceedings of the Conference on Algebraic Multilevel Iteration Methods with Applications*, University of Nijmegen, Nijmegen, The Netherlands, 1996.
- [3] O. AXELSSON AND P. S. VASSILEVSKI, *Algebraic multilevel preconditioning methods I*, Numer. Math., 56 (1989), pp. 157–177.
- [4] Z.-Z. BAI, *A class of hybrid algebraic multilevel preconditioning methods*, Appl. Numer. Math., 19 (1996), pp. 389–399.
- [5] R. E. BANK, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 7.0*, Frontiers Applied Math., Vol. 15, SIAM, Philadelphia, 1994.
- [6] ———, *Hierarchical bases and the finite element method*, in Acta Numerica 1996 (A. Iserles, ed.), Cambridge University Press, Cambridge, UK, 1996, pp. 1–43.
- [7] R. E. BANK, T. F. DUPONT, AND H. YSERENTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.
- [8] R. E. BANK AND R. K. SMITH, *General sparse elimination requires no permanent integer storage*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 574–584.
- [9] R. E. BANK AND C. WAGNER, *Multilevel ILU decomposition*, Numer. Math., (to appear).
- [10] R. E. BANK AND J. XU, *The hierarchical basis multigrid method and incomplete LU decomposition*, in Seventh International Symposium on Domain Decomposition Methods for Partial Differential Equations (D. Keyes and J. Xu, eds.), AMS, Providence, RI, 1994, pp. 163–173.
- [11] ———, *An algorithm for coarsening unstructured meshes*, Numer. Math., 73 (1996), pp. 1–36.
- [12] D. BRAESS, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393.
- [13] A. BRANDT, S. MCCORMICK, AND J. RUGE, *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations*, tech. report, Institute for Computational Studies, Colorado State University, Fort Collins CO, 1982.
- [14] ———, *Algebraic multigrid (amg) for sparse matrix equations*, in Sparsity and Its Applications (D. J. Evans, ed.), Cambridge University Press, Cambridge, UK, 1984.
- [15] J. E. DENDY, *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386.
- [16] S. C. EISENSTAT, M. C. GURSKY, M. SCHULTZ, AND A. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Statist. Comput., 2 (1982), pp. 225–237.
- [17] H. C. ELMAN AND X. ZHANG, *Algebraic analysis of the hierarchical basis preconditioner*, SIAM J. Matrix Anal., 16 (1995), pp. 192–206.
- [18] A. GEORGE AND J. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.
- [19] G. GOLUBOVICI AND C. POPA, *Interpolation and related coarsening techniques for the algebraic multigrid method*, in Multigrid Methods IV, Proceedings of the Fourth European Multigrid Conference, Amsterdam, 1993, vol. 116 of Internat. Schriftenreihe Numer. Math, Basel, 1994, Birkhäuser, pp. 201–213.
- [20] R. GUO AND R. D. SKEEL, *An algebraic hierarchical basis preconditioner*, Appl. Numer. Math., 9 (1992), pp. 21–32.
- [21] W. HACKBUSCH, *Multigrid Methods and Applications*, Springer-Verlag, Berlin, 1985.
- [22] W. HACKBUSCH AND G. WITTUM, *Incomplete Decompositions – Theory, Algorithms and Applications*, vol. 41 of Notes Numer. Fluid Mech., Vieweg, Braunschweig, 1993.
- [23] G. KARYPIS AND V. KUMAR, *Analysis of multilevel graph partitioning*, Tech. Report 95-037, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

- [24] ———, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Statist. Comput., (to appear).
- [25] D. J. ROSE, *A graph theoretic study of the numeric solution of sparse positive definite systems*, in Graph Theory and Computing, Academic Press, New York, 1972.
- [26] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers Applied Math., SIAM, Philadelphia, PA, 1987, pp. 73–130.
- [27] H. YSERENTANT, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.