ORDERING, ANISOTROPY AND FACTORED SPARSE APPROXIMATE INVERSES*

ROBERT BRIDSON† AND WEI-PAI TANG†

Abstract. We consider ordering techniques to improve the performance of factored sparse approximate inverse preconditioners, concentrating on the AINV technique of M. Benzi and M. Tůma. Several practical existing unweighted orderings are considered along with a new algorithm, Minimum Inverse Penalty (MIP), that we propose. We show how good orderings such as these can improve the speed of preconditioner computation dramatically, and also demonstrate a fast and fairly reliable way of testing how good an ordering is in this respect. Our test results also show that these orderings generally improve convergence of Krylov subspace solvers, but may have difficulties particularly for anisotropic problems. We then argue that weighted orderings, which take into account the numerical values in the matrix, will be necessary for such systems. After developing a simple heuristic for dealing with anisotropy we propose several practical algorithms to implement it. While these show promise, we conclude a better heuristic is required for robustness.

Key words. conjugate gradient-type methods, preconditioner, approximate inverse, ordering methods, anisotropy

AMS subject classifications. 65F10, 65F50

1. Introduction.

Consider solving the system of linear equations:

$$Ax = b$$

where A is a sparse $n \times n$ matrix. Depending on the size of A and the nature of the computing environment an iterative method, with some form of preconditioning to speed convergence, is a popular choice. Approximate inverse preconditioners, whose application requires only (easily parallelized) matrix-vector multiplication, are of particular interest today. Several methods of constructing approximate inverses have been proposed (e.g. [2, 3, 9, 20, 22, 24]), falling into two categories: those that directly form an approximation to A^{-1}, and those that form approximations to the inverses of the matrix's LU factors. This second category currently shows more promise than the first, for three reasons. First, it is easy to ensure that the factored preconditioner is non-singular, simply by making sure both factors have nonzero diagonals. Second, the factorization appears to allow more information per nonzero to be stored, improving convergence [4, 8]. Third, the set-up costs for creating these preconditioners can often be much less [4].

However, unlike A^{-1} itself, the inverse LU factors are critically dependent on the ordering of the rows and columns—indeed, they will not exist in general for some orderings. Even in the case of a SPD matrix, direct methods have shown how important ordering can be. Thus any factored approximate inverse scheme must handle ordering with thought. In particular, for an effective preconditioner an ordering that minimizes the size of the "dropped" entries is needed—decreasing the error between the approximate inverse factors and the true ones (see [14] for a discussion of this in the context of ILU).

In this paper we focus our attention on the AINV algorithm[3], which, via implicit Gaussian elimination with small-element dropping, constructs a factored approximate

* The work was supported by the Natural Sciences and Engineering Council of Canada, and Communications and Information Technology Ontario (CITO), funded by the Province of Ontario.

†Department of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada. email: rebridso@elora.uwaterloo.ca, wptang@elora.uwaterloo.ca

inverse:

$$A^{-1} \approx ZD^{-1}W^T$$

where Z and W are unit upper triangular, and D^{-1} is diagonal. However, the purely structural results presented here (in section 2) apply equally to other factored approximate inverse schemes. Whether the numerical results carry over is still to be determined. For example, conflicting evidence has been presented in [5] and [16] about the effect on FSAI[22], which perhaps will only be resolved when the issue of sparsity pattern selection for FSAI has been settled.

Some preliminary work in studying the effect of ordering on the performance of AINV has shown promising results[3] (a more recent work by the same authors is [5]). We carry this research forward in sections 2–3, realizing significant improvements in speed of preconditioner computation and observing some beneficial effects on convergence, but noting that structural information alone is not always enough. We then turn our attention to anisotropic problems. For the ILU class of preconditioners it has been determined that orderings which take the numerical values of the matrix into account are useful—even necessary (e.g. [10, 11, 12, 14]). Sections 4–6 try to answer the question of whether the same thing holds for factored approximate inverses. The appendix contains the details of our test results.

2. Unweighted Orderings.

Intuitively, the smaller the size of the dropped portion from the true inverse factors, the better the approximate inverse will be. We will for now assume that the magnitudes of the inverse factors' nonzeros are distributed roughly the same way under different orderings. (Our experience shows this is a fairly good assumption for typical isotropic problems, but as we shall see later, this breaks down for anisotropic matrices in particular) Then we can consider the simpler problem of reducing the *number* of dropped nonzeros, instead of their size. Of course, for sparsity we also want to retain as few nonzeros as possible; thus we really want to reduce the number of nonzeros in the exact inverse factors—a quantity we call inverse factor fill, or I_F fill.

Definition 2.1. *Let A be a square matrix with a triangular factorization $A = LU$. The I_F fill of A is defined to be the total number of nonzeros in the inverses of L and U, assuming no cancellation in the forming of those inverses.*

For simplicity we restrict our discussion to the symmetric positive-definite case, first examining I_F fill and then considering several existing ordering algorithms that may be of help. We finish the section by proposing a new ordering scheme, which we call MIP. The application to the unsymmetric case is straightforward.

The following discussion makes use of some concepts from graph theory. The graph of an $n \times n$ matrix A is a directed graph on n nodes labelled 1, …, n, with an arc $i \to j$ if and only if $A_{ij} \neq 0$. A directed path, or dipath, is an ordered set $\langle u_1, u_2, \ldots, u_k\rangle$ such that the arcs $u_1 \to u_2$, …, $u_{k-1} \to u_k$ all exist—often this is written as $u_1 \to \cdots \to u_k$. See chapter 3 of [17], for example, for further explanation.

From Gilbert[19] and Liu[23], we have the following graph theoretic characterization of the structure of the inverse Cholesky factor:

Theorem 2.2. *Let A be a SPD matrix with Cholesky factor L. Then assuming no cancellation the structure of $Z = L^{-T}$ corresponds to the transitive closure*[1] *of the*

[1]The transitive closure of a directed graph $\mathcal{G}$ is a graph $\mathcal{G}^*$ on the same vertices with an arc $u \to v$ for any vertices u and v that were connected by a dipath $u \to \cdots \to v$ in $\mathcal{G}$.

graph of L^T, that is, for $i < j$ we have $Z_{ij} \neq 0$ if and only if there is a dipath from i to j in the graph of L^T. Furthermore, this is the same structure as given by the transitive closure of the elimination tree of A (the transitive reduction[2] *of L^T).*

Notice that the last structure characterization simply means that for $i > j$, $Z_{ij} \neq 0$ if and only if j is an ancestor of i in the elimination tree. This allows us to significantly speed the computation of the preconditioner given a bushy elimination tree, as well as allowing for parallelism—for the calculation of column j in the factors, only the ancestor columns need be considered (a coarser-grain version of this parallelism via graph partitioning has been successfully implemented in [6] for example). Another product of this characterization is a simple way of computing the I_F fill of a SPD matrix, obtained by summing the number of non-zeros in each column of the inverse factor and multiplying by two for the other (transposed) factor:

THEOREM 2.3. *The I_F fill of a SPD matrix is simply twice the sum of the depths of all nodes in its elimination tree. In particular, the number of nonzeros in column j of the true inverse factor Z is given by the number of nodes in the subtree of the elimination tree rooted at j.*

These results suggest orderings that avoid long dipaths in L^T (i.e. paths in L^T with monotonically increasing node indices), as these cause lots of I_F fill, quadratic in their length. Alternatively, we are trying to get short and bushy elimination trees.

Another useful characterization of I_F fill using notions from [17, 18] allows us to do a cheap "inverse symbolic factorization"—determing the nonzero structure of the inverse factors—without using the elimination tree, which is essential for our Minimum Inverse Penalty ordering algorithm presented later.

THEOREM 2.4. *$Z_{ij} \neq 0$ if and only if i is reachable from j strictly through nodes eliminated previous to i—or in terms of the quotient graph model, if i is contained in a supernode adjacent to j at the moment when j is eliminated.*

Based on the heuristic and results above, we now examine several existing orderings which might do well and propose a new scheme to directly implement the heuristic of reducing I_F fill.

Red-Black. The simplest ordering we consider is (generalized) Red-Black, where a maximal independent set of ("red") nodes is ordered first, and the remaining ("black") nodes are ordered next according to their original sequence. In that initial red block, there are no non-trivial dipaths, hence no off-diagonal entries in Z.

Minimum Degree. Benzi and Tůma have observed that Minimum Degree is generally beneficial for AINV[3]. This is justified by noting that Minimum Degree typically substantially reduces the height of the elimination tree, hence should reduce I_F fill.

As an aside, notice that direct-method fill-reducing orderings do not necessarily reduce I_F fill. For example, a good envelope ordering will likely give rise to a very tall, narrow elimination tree—typically just a path—and thus give full inverse factors. Again, it should be noted that this isn't necessarily a bad thing if the inverse factors still have very small entries, but without using numerical information from the matrix

[2]The transitive reduction of a directed graph $\mathcal{G}$ is a graph $\mathcal{G}^\circ$ with the minimum number of arcs but still possessing the same transitive closure as $\mathcal{G}$.

our experience is that envelope orderings don't manage this. This may seem at variance with the result in [13] (elaborated in [5] for factored inverses) that for banded SPD matrices the rate of decay of the entries in the inverse has an upper bound that decreases as the bandwidth decreases. However, decay was measured there in terms of distance from the diagonal, which is really only suitable for small bandwidth orderings; the results presented in section 5 of [13], measuring decay in terms of the unweighted graph distance, should make theoretical progress possible. We will return to this issue in sections 4–6.

Nested Dissection (ND). On the other hand, by ordering vertex separators last Nested Dissection (ND) avoids any long monotonic dipaths, and hence a lot of I_F fill. Alternatively, in trying to balance the elimination tree it reduces the sum of depths.

Minimum Inverse Penalty (MIP). Above we noted that we can very cheaply compute the number of nonzeros in each column of Z within a symbolic factorization. This allows to propose a new ordering, MIP (Minimum Inverse Penalty), an analogue to Minimum Degree. Minimum Degree is built around a symbolic Cholesky factorization of the matrix, at each step selecting the node(s) of minimum penalty to eliminate. The penalty was originally taken to be the degree of the node in the partially eliminated graph; later algorithms have used other related quantities including the external degree and approximate upper bounds. In MIP we follow the same greedy strategy only we compute a penalty for node v based on $Zdeg_v$, the number of nonzeros in the column of Z were v to be ordered next—the degree in the *inverse* Cholesky factor instead of the Cholesky factor—as well as on $Udeg_v$, the number of uneliminated neighbours of node v at the current stage of factorization (not counting supernodes). In our experiments we found the function $Penalty_v = 2Zdeg_v + Udeg_v$ to be fairly effective. Further research into a better penalty function is needed. Also, ideas from Minimum Degree such as multiple elimination, element absorption, etc. might be suitable here.

3. Testing Unweighted Orderings.

We used the symmetric part of the matrix for all the orderings. Red-Black was implemented with the straight-forward greedy algorithm to select a maximal independent set. Our Minimum Degree algorithm was AMDBAR, a top-notch variant due to Amestoy, Davis, and Duff [1]. We wrote our own ND algorithm that constructs vertex separators from edge separators given by a multi-level bisection algorithm. This algorithm coarsens the graph with degree-1 node compression and heavy-edge matchings until there are less than 100 nodes, bisects the small graph spectrally according to the Fiedler vector[15], and uses a greedy boundary-layer sweep to smooth in projecting back to the original. See [21], for example, for more details on this point.

The appendix provides further details about our testing. The tables contain data for both the unweighted orderings above and their weighted counterparts presented below—ignore the lower numbers for now. In brief, we selected several matrices from the Harwell-Boeing collection and tested them all with each ordering scheme. Table A.1 gives the true I_F fill for each matrix (or its symmetric part) and ordering. Tables A.2–A.4 give the preconditioner performance. As the number of nonzeros allowed in the preconditioner can have a significant effect on results, we standardized all our test runs: in each box the left number is a report for when the preconditioner had as many nonzeros as the matrix, and the right number for when the preconditioner had twice as many nonzeros.

FIG. 3.1. *Correlation of I_F fill and preconditioner computing time, normalized with respect to the given (original) ordering.*

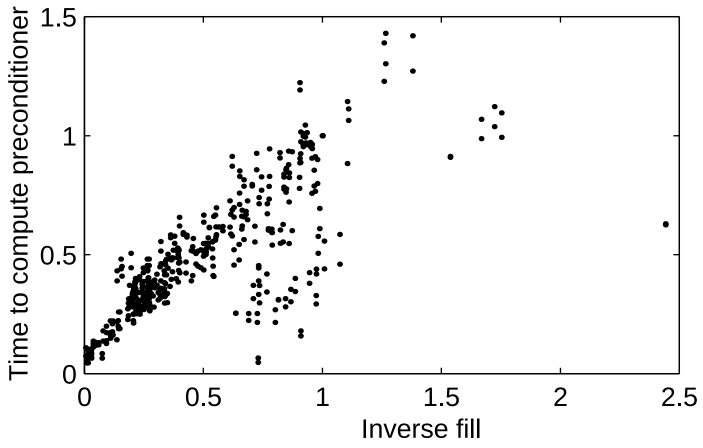


TABLE 3.1
Average decrease in number of iterations over the test set, in percentages of the iterations taken by the given ordering.

Fill	Red-Black	AMDBAR	ND	MIP
1	94%	75%	76%	75%
2	93%	73%	83%	70%

In terms of I_F fill reduction, AMDBAR, ND, and MIP are always the best three by a considerable factor. ND wins 15 times, AMDBAR 5 times, and MIP 3 times, with one tie. Red-Black beats the natural ordering, but not dramatically.

It is clear that ordering can help immensely for accelerating the computation of the preconditioner. ND is the winner, followed by AMDBAR, MIP, and then Red-Black quite a bit behind. The preconditioner computing time is closely correlated to I_F fill—see figure 3.1. Thus calculating I_F fill provides a fast and reasonably good test to indicate how efficient an (unweighted) ordering is for preconditioner computation—perhaps not an important point if the iteration time dominates the set-up time, but this may be useful for applications where the reverse is true.

The effect of ordering on speed of solution is less obvious. The poor behaviour of PORES2[3], SHERMAN2, and WATSON5 indicate that AINV probably isn't appropriate (although if we had properly treated SHERMAN2 as a block matrix instead it might have gone better). Notice in particular that sometimes lowering the drop tolerance, increasing the size of the preconditioner and hopefully making it more accurate, actually degrades convergence for these indefinite problems. From the remaining matrices, we compared the average decrease in number of iterations over the given ordering—see table 3.1. Particularly given its problems with SAYLR4, WATSON4, WATT1 and WATT2 Red-Black cannot be viewed as a good ordering. MIP is overall the best, although it had a problem with WATT1, whereas the close contender AMDBAR did fairly well on all but the difficult three mentioned above. The good

[3]In [3] somewhat better convergence was achieved for PORES2, presumably due to implementation differences in the preconditioner or its application.

I_F fill reducing orderings all made worthwhile improvements in convergence rates, but it is surprising that ND did the least considering its superior I_F fill reduction. Clearly our intuition that having less nonzeros to drop makes a better preconditioner has merit, but doesn't tell the entire story.

4. Anisotropy.

In the preceding test results we find several exceptions to the general rule that AMDBAR, ND, and MIP perform similarly, even ignoring PORES2, SHERMAN2, and WATSON5. For example, there are considerable variations for each of ALE3D, BCSSTK14, NASA1824, ORSREG1, SAYLR4, and WATSON4. More importantly, these variations are not correlated with I_F fill; some other factor is at work. Noticing that each of these matrices are quite anisotropic, and recalling the problems anisotropy poses for ILU, we are led to investigate weighted orderings.

We first develop a heuristic for handling anisotropic matrices. The goal in mind is to order using the differing strengths of connections to reduce the magnitude of the inverse factor entries. Then even if we end up with more I_F fill (and hence drop more nonzeros), the magnitude of the discarded portion of the inverse factors may be smaller and give a more accurate preconditioner.

Again, we only look at symmetric positive-definite A. Let $A = LDL^T$ be the modified Cholesky factorization, where L is unit lower triangular and D is diagonal. Then $Z = L^{-T}$, and since $(I - L^T)$ is zero on and below the diagonal hence nilpotent, we have:

$$Z = I + (I - L^T) + (I - L^T)^2 + \ldots + (I - L^T)^{n-1}$$

Then for $i \leq j$:

$$Z_{i,j} = \sum_{i=i_1<i_2<\cdots<i_p=j} (-1)^{p-1} L_{i_2,i_1} L_{i_3,i_2} \cdots L_{i_p,i_{p-1}}$$

The nonzero entries in this sum correspond to the monotonically increasing dipaths $i = i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_p = j$ in L^T. Our orderings should therefore avoid having many such dipaths which involve large entries of L, as each one could substantially increase the magnitude of Z's entries. Thus we want to move the large entries away from the diagonal, so they cannot appear in many monotonic dipaths. In other words, after a node has been ordered, we want to order so that its remaining neighbours with strong L-connections come as late as possible after it.

For the purposes of our ordering heuristic, we want an easy approximation to L independent of the eventual ordering chosen—something that can capture the order of magnitude of entries in L but doesn't require us to decide the ordering ahead of time. Assuming A has an adequately dominant diagonal without too much variation, we can take the absolute value of the lower triangular part of A, symmetrically rescaled to have a unit diagonal. (This can be thought of as a scaled Gauss-Seidel approximation) Our general heuristic is then to delay strong connections in this approximating matrix M defined by:

$$M = F^{-1/2}|A|F^{-1/2} \quad \text{where} \quad F = diag|A|$$

An alternative justification of this heuristic, simply in the context of reducing the magnitude of entries in L, is presented in [10].

Now consider a simple demonstration problem to determine whether this heuristic could help. The matrix SINGLEANISO comes from a 5-point finite difference

FIG. 4.1. *The two orderings of SINGLEANISO, depicted on the domain. Lighter shaded boxes indicate nodes ordered later.*

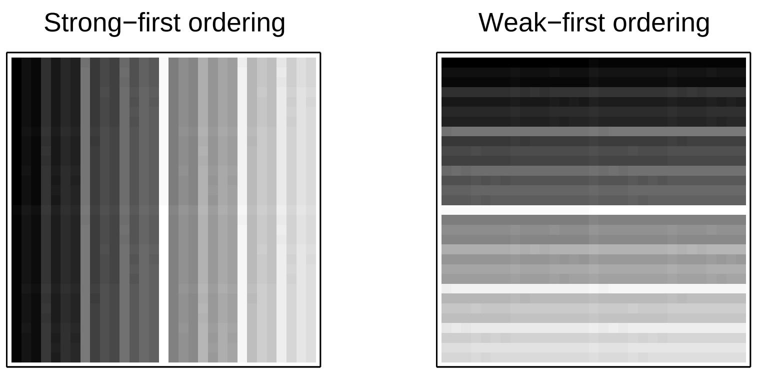


TABLE 4.1
Performance of strong-first ordering versus weak-first ordering for SINGLEANISO.

Ordering	Time to compute preconditioner	Number of iterations	Time for iterations
Strong-first	0.51	29	0.4
Weak-first	0.38	25	0.25

discretization on a regular 31×31 grid of the following PDE:

$$u_{xx} + 1000u_{yy} = F$$

Here the edges of A (and M) corresponding to the y-direction are 1000 times heavier than those corresponding to the x-direction. We try comparing two I_F fill reducing orderings. The first ordering ("Strong-first") block-orders the grid columns with nested dissection, and then internally orders each block with nested dissection—this brings the strong connections close to the diagonal. The second ("Weak-first") block-orders the grid rows instead, pushing the strong connections away from the diagonal, delaying them until the last. These are illustrated in figure 4.1 where each square of the grid is shaded according to its place in the ordering.

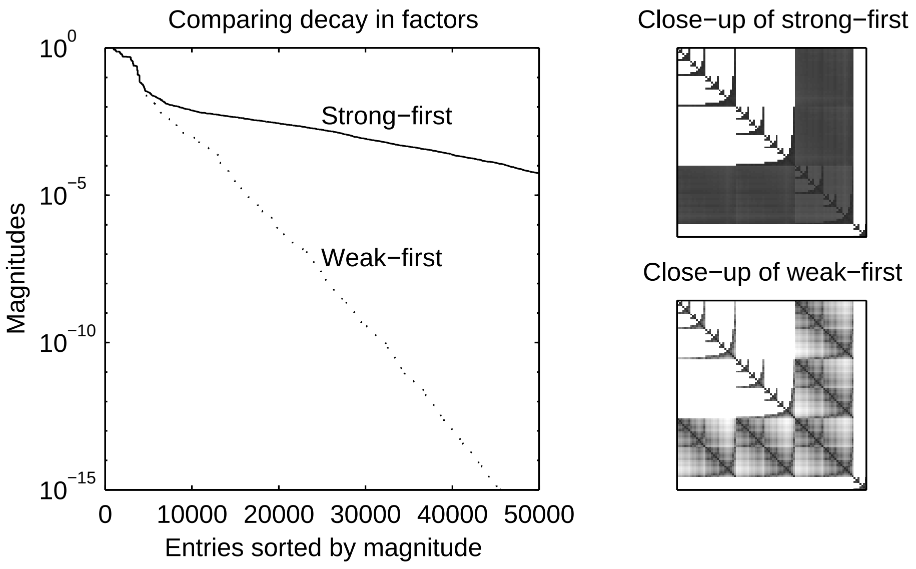
Both orderings produce a reasonable I_F fill of 103,682, with isomorphic elimination trees. However, they give very different performance at the first level of fill—see table 4.1. In all respects the weak-first ordering is significantly better than the strong-first one.

In figure 4.2 we plot the decay of the entries in the inverse factors resulting from the two orderings, and show parts of those factors. The much smaller entries from the weak-first ordering confirm our heuristic.

5. Weighted Orderings.

In our experience it appears that I_F fill reduction typically helps to also reduce the magnitude of entries in the inverse factors, but blind as it is to the numerical values in the matrix it can make mistakes such as allowing strong connections close to the diagonal. In creating algorithms for ordering general matrices, we thus have tried to simply modify the unweighted algorithms to consider the numerical values.

FIG. 4.2. *Comparison of the magnitude of entries in inverse factors for the different orderings of SINGLEANISO. The close-up images of the actual factors are shaded according to the magnitude of the non-zeros—darker means bigger.*



Weighted Nested Dissection (WND). Consider the spectral bipartition algorithm. Finding the Fiedler vector, the eigenvector of the Laplacian of the graph with second smallest eigenvalue (see [15]), is equivalent to minimizing (over a space orthogonal to the constant vectors):

$$\sum_{(i,j) \text{ is an edge}} (x_i - x_j)^2$$

We then make the bipartition depending on which side of the median each entry lies. Notice that the closer together two entries are in value—i.e. the smaller $(x_i - x_j)^2$ is—the more likely those nodes will be ordered on the same side of the cut. We would like weakly connected nodes (where M_{ij} is small) to be in the same part and the strong connections to be in the edge cut, so we try minimizing the following weighted quadratic sum:

$$\sum_{(i,j) \text{ is an edge}} \frac{(x_i - x_j)^2}{M_{ij}}$$

where M is the scaled matrix mentioned above. This corresponds to finding the eigenvector with second smallest eigenvalue of the weighted Laplacian matrix for the graph defined by:

$$(i,j) \text{ is an edge if and only if } M_{ij} \neq 0, \qquad weight(i,j) = 1/M_{ij}$$

Thus we modify ND simply by changing the Laplacian used in the bipartition step to this weighted Laplacian. Fortunately our multi-level approach with heavy-edge matchings typically will eliminate the largest off-diagonal entries as well as substantially decreasing the size of the eigenproblem, making it easy to solve, so our Weighted Nested Dissection (WND) is very reasonable to compute.

OutIn. Our other weighted orderings are based on an intuition from finite-difference matrices. We expect that the nodes most involved in long, heavy paths are those near the weighted centre of the graph (those with minimum weighted eccentricity). The nodes least involved in paths are intuitively the ones on the weighted periphery of the graph. Thus OutIn orders the periphery first, and proceeds to an approximate weighted centre ("from the outside to the inside"). To efficiently find an approximate weighted centre we use an iterative algorithm:

Algorithm 1 (Approximate Weighted Centre).

```
• Set $r \leftarrow \frac{1}{2}$, $i \leftarrow 1$, $v_1 \leftarrow$ some random node.
• Do
    – Calculate the $M$-weighted shortest paths and $M$-weighted distances to
      other nodes from $v_i$ (where the distance is the minimum sum of weights,
      given by $M$, along a connecting path).
    – Select a node $e_i$ of maximum distance from $v_i$.
    – Travel $r$ of the way along a shortest path from $v_i$ to $e_i$, saving the re-
      sulting node as $v_{i+1}$.
    – Set $r \leftarrow \frac{r}{2}$, $i \leftarrow i + 1$.
• End loop when $v_i = v_{i-1}$ and return $v_i$ as the approximate centre.
```

Then our OutIn ordering is:

Algorithm 2 (OutIn).

```
• Compute $M$.
• Get an approximate weighted centre $c$ for $M$.
• Calculate the distances and shortest paths from all other nodes in $M$ to $c$.
• Return the nodes in sorted order, with most distant first and $c$ last.
```

Despite our earlier remark that envelope orderings might not be useful, the weight information actually lets OutIn perform significantly better than the natural ordering, by reducing the size of the nonzeros in the factors if not the number. However, why not combine OutIn with an I_F fill reducing ordering to try to get the best of both worlds? We thus test OutIn as a preprocessing stage before applying Red-Black, Minimum Degree, or MIP. We note that the use of hash tables and other methods to accelerate the latter two means that it's not true that the precedence set by OutIn will always be followed in breaking ties for minimum penalty.

As an aside, we also considered modifying Minimum Degree and MIP with tie-breaking directly based on the weight of a candidate node's connections to previously selected nodes. Weighted tie-breaking (with RCM) has proved useful before in the context of ILU[11]. However, for the significant extra cost incurred by this tie-breaking, this achieved little here—it appears that a more global view of weights is required when doing approximate inverses.

Before proceeding to our large test set, we verify that these orderings are behaving as expected with another demonstration matrix. ANISO is a similar problem to SINGLEANISO only with four abutting regions of anisotropy with differing directions[12]—see figure 5.1 for a diagram showing the directions in the domain. As shown in table 5.1, the results for WND over ND and OutIn/MIP over MIP didn't change, but there was a significant improvement in the other orderings.

6. Testing Weighted Results. We repeated the tests for our weighted orderings, with results given in tables A.1–A.4. For unsymmetric matrices, we used $|A| + |A|^T$ to define M in WND and OutIn, avoiding the issue of directed edges as

FIG. 5.1. *Schematic showing the domain of ANISO. The arrows indicate the direction of the strong connections.*

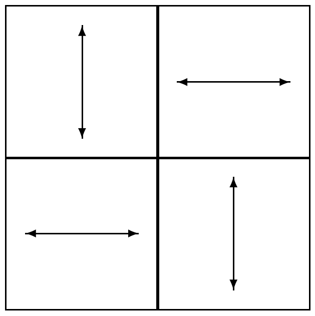


TABLE 5.1
Performance of weighted orderings versus unweighted orderings for ANISO.

Ordering	I_F fill	Time to compute preconditioner	Number of iterations	Time for iterations
Given	462	0.41	118	1.2
OutIn	266	0.31	77	0.76
Red-Black	239	0.28	107	1.13
OutIn/RB	208	0.28	61	0.61
AMDBAR	69	0.14	66	0.65
OutIn/AMD	69	0.15	48	0.5
ND	72	0.14	65	0.64
WND	84	0.15	65	0.65
MIP	83	0.16	47	0.49
OutIn/MIP	93	0.18	47	0.48

with unweighted orderings. In each box of the tables, the lower numbers correspond to the weighted orderings; we have grouped them with the corresponding unweighted orderings for comparison.

Only ND suffered in preconditioner computing time—our spectral weighting appears to be too severe, creating too much I_F fill. However, it is important to note that the increase in time is much less than that suggested by I_F fill—indeed, although WND gave several times more I_F fill for ADD32, MEMPLUS, SAYLR4, SHERMAN4, and WANG1, it actually allowed for slightly faster preconditioner computation. This verifies the merit of our heuristic. Both the natural ordering and Red-Black benefited substantially from OutIn in terms of preconditioner computation, and AMDBAR and MIP didn't seem to be effected very much—this could quite well be a result of the data structure algorithms which do not necessarily preserve the initial precedence set by OutIn.

In terms of improving convergence, we didn't fix the problems with PORES2, SHERMAN2, and WATSON5. These matrices have very weak diagonals anyhow, so our heuristics probably don't apply. OutIn and OutIn/RB are a definite improvement on the natural ordering and Red-Black, apart from on BCSSTK14 and WATSON4. The effect of OutIn on AMDBAR and MIP is not clear; usually there's little effect, and on some matrices (e.g. ALE3D and SAYLR4) it has an opposite effect on the two. WND shows more promise, improving convergence over ND considerably for ALE3D, BCSSTK14, ORSIRR1, ORSREG1, and SAYLR4. Its much poorer I_F fill reduction (generally by a factor of 4) gave it problems on a few matrices though.

7. Conclusions. It is clear that I_F fill reduction is crucial to the speed of preconditioner computation, often making an order of magnitude difference. We also saw that the I_F fill of a matrix can be computed very cheaply and gives a good indication of the preconditioner computation time, for unweighted orderings at least.

Reducing I_F fill typically also gives a more effective preconditioner, accelerating convergence—not only are the number of nonzeros in the true inverse factors decreased, but the magnitude of the portion that is dropped by AINV is reduced too. However, although ND gave the best I_F fill reduction, MIP gave the best acceleration so care must be taken. It would be interesting to determine why this is so. Probably several steps of Nested Dissection followed by MIP or a Minimum Degree variant on the subgraphs will prove to be the most practical ordering.

Anisotropy can have a significant effect on performance, both in terms of preconditioner computing time and solution time. Our Weighted Nested Dissection (WND) algorithm shows the most promise for a high-performance algorithm that can exploit anisotropy, perhaps after some tuning of the weights in the Laplacian matrix used. Robustness is still an issue; we believe a more sophisticated weighting heuristic is necessary for further progress.

Appendix. Testing Data.

Our test platform was a 180MHz Pentium Pro running Windows/NT. We used MATLAB 5.1, with the algorithm for AINV written as a MEX extension in C. Our AINV algorithm was a left-looking, column-by-column version, with off-diagonal entries dropped when their magnitude is below a user-supplied tolerance, and with the entries of D shifted to $\pm 10^{-3} \max |A|$ when their computed magnitude fell below $10^{-1}\epsilon \max |A|$. We also make crucial use of the elimination tree; in making a column conjugate with the previous columns, we only consider its descendents in the elimination tree (the only columns that could possibly contribute anything). This accelerates AINV considerably for low I_F fill orderings—e.g. SHERMAN3 with AMDBAR ordering and a drop tolerance of 0.1 is accelerated by a factor of four! An upcoming paper[7] will explore this more thoroughly.

To compare the orderings we selected several matrices, mostly from the Harwell-Boeing collection. First we found the amount of true I_F fill caused by each ordering, given in table A.1. We then determined drop-tolerances for AINV to produce preconditioners with approximately N and $2N$ nonzeros, where N is the number of nonzeros in the given matrix. For each matrix, ordering, and fill level we attempted to solve $Ax = b$ using BiCGStab (CG for s.p.d. matrices), where b was chosen so that the correct x is the vector of all 1's. Tables A.2, A.3, and A.4 give the CPU time taken for preconditioner computation, the iterations required to reduce the residual norm by a factor of 10^{-9}, and the CPU time taken by the iterations. We halted after 1800 iterations; the daggers in tables A.3 and A.4 indicate no convergence at that point.

In each box of the tables, the upper line corresponds to the unweighted ordering and the lower line its weighted counterpart. In tables A.2, A.3, and A.4 the numbers on the left of the box correspond to the low fill tests and those on the right to the high fill tests.

To highlight the winning ordering for each matrix, we have put the best numbers in boldface underlined.

Table A.1

Comparison of I_F fill caused by different orderings. *Nonzero counts are given in thousands of nonzeros. In each box the upper number corresponds to the unweighted ordering, and the lower number to its weighted counterpart.*

Name	n	NNZ	Given OutIn	Red-Black OutIn/*	AMDBAR OutIn/*	ND WND	MIP OutIn/*
ADD20	2395	13	2660	2228	119	156	**97**
			244	243	119	1689	**97**
ADD32	4960	20	9083	8662	191	134	**52**
			1062	706	154	669	53
ALE3D	1590	45	365	561	241	**197**	404
			628	608	283	639	329
BCSSTK14	1806	63	1554	1446	432	**402**	573
			1486	1506	419	1526	709
MEMPLUS	17758	99	156022	119739	1618	2362	1736
			4468	4435	**1615**	113777	1736
NASA1824	1824	39	1664	1506	**358**	452	489
			1408	1411	374	1572	454
NASA2146	2146	72	2304	2153	623	**510**	698
			2116	2114	644	2242	808
ORSIRR1	1030	6.9	458	284	154	**133**	206
			394	328	153	462	208
ORSIRR2	886	6.0	320	197	**115**	126	123
			354	279	101	343	174
ORSREG1	2205	14	2432	1414	510	**502**	643
			2092	1527	512	1913	879
PORES2	1224	9.6	738	606	237	202	251
			571	541	**179**	720	317
PORES3	532	3.5	107	83	42	**20**	23
			77	70	40	76	**20**
SAAD100	8000	54	32004	16770	6878	**6238**	12095
			28974	19841	6815	26054	12750
SAYLR4	3564	22	6352	3519	1275	**1153**	2182
			3236	2311	1306	5091	1650
SHERMAN1	1000	3.8	224	123	55	**52**	61
			150	112	59	164	72
SHERMAN2	1080	23	551	500	269	**262**	357
			539	531	275	544	346
SHERMAN3	5005	20	3708	2090	749	**746**	1464
			3437	2414	751	3212	1451
SHERMAN4	1104	3.8	149	102	40	40	40
			128	91	**37**	114	59
SHERMAN5	3312	21	1340	1122	414	**334**	465
			1117	1103	424	1186	451
SWANG1	3169	21	3865	3666	546	**424**	941
			3513	2722	525	3513	837
WANG1	2903	19	4215	2262	**1056**	1122	1795
			3133	2327	1076	3557	1684
WATSON4	467	2.8	72	69	11	13	**7.8**
			57	39	10	53	8.5
WATSON5	1853	9.6	341	471	**46**	141	67
			432	430	54	833	52
WATT1	1856	11	1719	891	400	**351**	576
			1165	878	425	1184	466
WATT2	1856	12	1723	895	401	**353**	583
			1141	806	496	1251	472

Table A.2
Comparison of CPU time for preconditioner computation. *In each box the upper numbers correspond to the unweighted ordering, and the lower numbers to its weighted counterpart. The numbers on the left refer to the low-fill test, and the numbers on the right to the high-fill test.*

	Given		Red-Black		AMDBAR		ND		MIP	
Name	**OutIn**		**OutIn/***		**OutIn/***		**WND**		**OutIn/***	
ADD20	9.8	12.8	8.1	10.7	1.3	1.5	1.3	1.6	**1.2**	**1.4**
	1.7	1.6	2.0	1.8	1.3	1.5	2.5	3.2	1.3	1.5
ADD32	33.9	45.6	32.7	43.5	3.4	3.6	3.2	3.4	3.7	3.4
	7.1	8.0	6.1	6.3	3.5	3.4	**2.9**	**3.0**	3.7	3.4
ALE3D	14.3	26.6	13.0	24.1	9.4	16.1	**6.4**	**11.0**	15.9	28.3
	16.0	27.6	15.3	26.2	11.2	19.5	15.7	26.4	11.8	20.7
BCSSTK14	6.2	11.1	6.0	10.7	2.3	3.7	**2.0**	**3.3**	3.0	4.7
	5.6	8.4	5.6	8.5	2.2	3.6	3.6	5.6	3.5	5.7
MEMPLUS	832	1174	594	789	58.6	64.5	59.4	53.8	**54.0**	**52.1**
	70.6	76.6	81.0	84.5	61.6	61.9	54.7	55.8	65.1	58.2
NASA1824	4.6	7.8	4.0	7.1	**1.3**	**2.1**	1.5	2.3	1.8	2.8
	3.9	6.0	3.9	6.1	1.4	2.2	1.9	3.0	1.7	2.8
NASA2146	8.4	15.2	8.5	14.7	3.0	5.0	**2.4**	**3.8**	3.5	5.7
	8.4	14.5	8.5	14.7	3.1	5.1	2.8	4.5	4.2	6.8
ORSIRR1	1.8	3.2	1.2	1.8	0.7	1.0	**0.7**	**0.9**	0.9	1.2
	1.3	1.7	1.1	1.8	0.8	1.1	1.0	1.4	0.9	1.3
ORSIRR2	1.2	2.1	0.8	1.2	0.6	0.8	0.6	0.8	0.6	0.8
	1.4	1.9	1.1	1.3	**0.5**	**0.7**	0.7	1.0	0.8	0.9
ORSREG1	7.6	10.7	4.7	6.4	2.8	3.9	**2.7**	**3.8**	3.3	4.6
	6.4	8.8	5.3	7.1	2.9	3.6	4.6	6.4	4.4	6.1
PORES2	2.8	4.0	2.5	3.8	1.3	1.9	**0.7**	**1.1**	1.3	1.8
	1.7	2.5	2.0	2.9	1.0	1.5	1.2	1.8	1.6	2.4
PORES3	0.4	0.5	0.3	0.5	0.2	0.3	**0.1**	**0.2**	**0.1**	**0.2**
	0.3	0.5	0.3	0.5	0.2	0.3	**0.1**	**0.2**	**0.1**	**0.2**
SAAD100	134	196	82.1	120	47.4	65.8	**39.4**	**56.4**	77.3	107
	160	240	117	179	51.0	70.6	41.6	60.7	88.0	122
SAYLR4	8.5	12.1	5.9	7.1	2.8	3.8	2.3	2.8	4.1	5.0
	4.6	6.1	4.1	4.8	2.9	3.5	**2.3**	**2.6**	3.3	3.9
SHERMAN1	0.3	0.3	0.2	0.2	**0.1**	**0.1**	**0.1**	**0.1**	0.1	0.2
	0.2	0.3	0.2	0.2	0.1	0.2	0.1	0.2	0.2	0.2
SHERMAN2	5.8	11.1	5.4	9.9	3.0	5.0	3.0	5.0	3.2	5.3
	5.2	8.9	5.0	8.8	3.2	5.5	4.0	6.8	3.0	5.1
SHERMAN3	18.1	27.8	11.2	17.1	6.1	8.0	**6.0**	**7.8**	9.5	12.9
	18.9	27.6	13.8	19.8	6.2	8.4	6.4	8.4	9.6	13.5
SHERMAN4	0.6	1.0	0.4	0.6	0.2	0.4	0.3	0.4	**0.2**	**0.3**
	0.6	0.9	0.4	0.6	**0.2**	**0.3**	0.3	0.3	0.3	0.4
SHERMAN5	12.1	21.2	9.5	16.4	4.1	6.6	**3.6**	**5.7**	4.1	6.3
	7.6	11.7	7.3	11.6	4.3	6.7	4.8	7.3	4.1	6.3
SWANG1	18.7	29.4	18.2	28.0	4.2	5.8	**3.2**	**4.4**	6.4	9.2
	18.2	29.8	14.8	23.3	4.1	4.2	3.4	4.6	5.9	8.5
WANG1	19.6	31.3	10.8	16.4	**6.2**	**8.8**	6.2	8.9	9.2	13.2
	16.2	24.2	12.1	18.0	6.4	9.2	**6.2**	**8.8**	9.2	13.3
WATSON4	0.3	0.4	0.3	0.4	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
	0.2	0.2	0.2	0.2	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
WATSON5	1.6	2.0	2.1	2.8	**0.7**	**0.8**	1.0	1.2	0.8	0.9
	2.1	2.9	2.0	2.8	**0.7**	**0.8**	1.0	1.2	0.8	0.9
WATT1	7.2	11.3	3.8	6.0	2.0	3.0	**1.8**	**2.5**	2.7	4.0
	4.8	7.7	3.8	5.8	2.2	3.2	**1.8**	**2.5**	2.4	3.6
WATT2	7.4	12.0	4.2	6.6	2.1	3.0	**1.8**	**2.6**	2.8	3.9
	5.1	7.5	3.7	5.5	2.7	3.9	1.9	2.6	2.5	3.5

Table A.3

Comparison of iterations required to reduce residual norm by 10^{-9}. *In each box the upper numbers correspond to the unweighted ordering, and the lower numbers to its weighted counterpart. The numbers on the left refer to the low-fill test, and the numbers on the right to the high-fill test.*

Name	Given OutIn		Red-Black OutIn/*		AMDBAR OutIn/*		ND WND		MIP OutIn/*	
ADD20	8	5	8	5	7	4	8	4	**8**	**3**
	8	4	8	4	8	4	8	4	**8**	**3**
ADD32	6	4	6	4	**5**	**2**	5	3	**5**	**2**
	6	4	6	3	**5**	**2**	6	3	**5**	**2**
ALE3D	126	60	70	48	37	44	68	45	33	25
	33	29	24	27	**28**	**21**	34	35	41	39
BCSSTK14	77	57	76	56	71	47	107	97	**64**	**44**
	109	99	109	105	69	48	85	66	63	46
MEMPLUS	135	13	108	7	18	5	17	6	**21**	**3**
	19	9	18	9	18	5	19	6	**21**	**3**
NASA1824	945	727	915	734	837	511	**708**	**498**	850	496
	917	801	920	821	883	601	822	622	834	584
NASA2146	76	56	85	70	57	34	**58**	**30**	52	37
	85	40	80	40	67	36	69	33	57	33
ORSIRR1	38	22	40	20	32	20	39	23	34	18
	33	20	**31**	**17**	31	19	33	18	32	19
ORSIRR2	39	28	37	21	33	24	31	20	33	19
	38	20	32	20	**31**	**19**	34	17	34	22
ORSREG1	43	26	41	19	40	18	39	28	42	19
	39	20	**37**	**16**	39	20	35	18	41	21
PORES2	160	762	**180**	**99**	716	448	†	1120	1117	189
	†	1108	754	409	†	315	†	†	662	630
PORES3	87	28	90	23	81	31	36	21	35	20
	51	35	51	38	95	33	37	17	**34**	**18**
SAAD100	25	19	19	13	**17**	**12**	19	16	**17**	**12**
	21	17	17	14	**17**	**12**	19	13	**17**	**12**
SAYLR4	1386	70	70	95	90	64	173	63	158	62
	1251	70	124	67	118	64	**92**	**62**	93	62
SHERMAN1	50	34	47	38	51	30	53	35	48	33
	56	41	57	35	**47**	**30**	46	33	49	32
SHERMAN2	†	280	**285**	**173**	†	†	†	1092	†	†
	†	†	†	†	†	†	†	†	†	†
SHERMAN3	245	98	111	81	111	70	114	81	**105**	**72**
	114	88	118	116	108	74	115	83	115	100
SHERMAN4	37	25	36	22	35	22	36	23	**34**	**22**
	37	26	36	23	36	22	35	23	36	22
SHERMAN5	34	24	30	23	31	19	**28**	**19**	28	21
	33	20	33	20	30	19	32	19	**28**	**19**
SWANG1	4	3	4	3	4	3	4	3	4	3
	4	3	4	3	4	3	4	3	4	3
WANG1	43	35	**38**	**27**	40	27	38	29	39	30
	42	32	40	28	39	26	40	29	**38**	**27**
WATSON4	32	10	39	11	**19**	**3**	28	6	28	4
	63	11	46	8	29	4	35	6	17	4
WATSON5	**306**	**611**	†	†	†	96	†	1022	†	290
	†	†	†	†	†	119	†	1056	†	274
WATT1	7	6	12	5	8	5	**6**	**5**	12	6
	7	5	12	9	12	5	12	4	9	9
WATT2	77	54	114	54	8	7	8	8	7	5
	49	5	8	5	14	5	**6**	**5**	**6**	**5**

Table A.4
Comparison of time taken for iterations. *In each box the upper numbers correspond to the unweighted ordering, and the lower numbers to its weighted counterpart. The numbers on the left refer to the low-fill test, and the numbers on the right to the high-fill test.*

	Given		**Red-Black**		**AMDBAR**		**ND**		**MIP**	
Name	**OutIn**		**OutIn/***		**OutIn/***		**WND**		**OutIn/***	
ADD20	0.4	0.4	0.4	0.4	0.4	0.3	0.4	0.3	0.4	0.2
	0.4	0.3	0.4	0.3	0.4	0.3	0.4	0.3	**0.4**	**0.2**
ADD32	0.6	0.5	0.6	0.5	**0.5**	**0.2**	0.5	0.4	**0.5**	**0.2**
	0.6	0.5	0.6	0.4	**0.5**	**0.2**	0.6	0.4	**0.5**	**0.2**
ALE3D	18.4	11.5	10.2	9.3	5.5	8.6	10.0	8.7	4.9	5.0
	4.9	5.8	3.6	5.4	**4.2**	**4.2**	5.0	6.9	6.1	7.7
BCSSTK14	9.3	9.0	9.2	8.9	8.6	7.5	13.0	15.4	**7.9**	**7.1**
	13.4	16.0	13.2	17.0	8.3	7.7	10.3	10.5	7.7	7.5
MEMPLUS	67.7	8.9	61.0	8.0	**9.7**	**5.0**	9.7	10.1	11.8	4.7
	14.7	11.4	10.4	6.8	22.0	5.7	15.0	4.3	11.1	4.5
NASA1824	71.4	73.5	68.2	73.5	63.3	52.1	**53.2**	**50.4**	65.3	52.0
	68.8	81.7	69.3	83.6	66.2	61.2	61.6	62.8	63.1	59.8
NASA2146	10.3	10.2	11.6	12.8	7.8	6.3	**7.9**	**5.6**	7.2	6.9
	11.6	7.4	10.9	7.4	9.2	6.7	9.4	6.0	7.9	6.1
ORSIRR1	1.0	0.7	1.1	0.7	0.8	0.7	1.0	0.8	0.9	0.6
	0.9	0.7	**0.8**	**0.6**	**0.8**	**0.6**	0.9	0.6	0.8	0.7
ORSIRR2	0.9	0.8	0.8	0.6	0.8	0.7	0.7	0.6	0.8	0.6
	0.9	0.6	0.7	0.6	**0.7**	**0.5**	0.8	0.5	0.8	0.6
ORSREG1	2.4	1.9	2.3	1.4	2.3	1.3	2.2	2.0	2.4	1.4
	2.2	1.5	2.1	1.2	2.2	1.5	**2.0**	**1.3**	2.3	1.6
PORES2	5.8	36.8	**6.8**	**4.9**	25.1	19.8	†	51.1	39.6	8.4
	†	50.6	26.8	18.6	†	14.2	†	†	23.2	28.2
PORES3	1.1	0.5	1.2	0.4	1.1	0.5	0.5	0.3	**0.4**	**0.3**
	0.7	0.6	0.7	0.6	1.3	0.5	0.5	0.3	**0.4**	**0.3**
SAAD100	5.9	5.7	4.5	3.9	**4.0**	**3.8**	4.3	4.6	**4.0**	**3.8**
	4.8	5.0	4.0	4.4	4.0	3.9	4.3	3.7	4.1	3.8
SAYLR4	71.8	4.6	3.9	6.4	4.7	4.6	8.9	4.3	8.3	4.3
	65.6	4.8	6.8	4.6	6.2	4.4	**4.8**	**4.3**	4.9	4.3
SHERMAN1	0.4	0.4	0.4	0.4	**0.4**	**0.3**	0.4	0.4	**0.4**	**0.3**
	0.5	0.4	0.5	0.4	**0.4**	**0.3**	**0.4**	**0.3**	**0.4**	**0.3**
SHERMAN2	†	29.8	**23.0**	**18.4**	†	†	†	115	†	†
	†	†	†	†	†	†	†	†	†	†
SHERMAN3	23.1	11.5	10.5	10.0	10.5	8.4	10.7	9.5	**9.9**	**8.7**
	10.8	10.3	11.2	13.8	10.3	8.8	10.8	9.8	10.9	12.2
SHERMAN4	0.7	0.6	**0.6**	**0.5**	0.6	0.6	**0.6**	**0.5**	**0.6**	**0.5**
	0.7	0.5	**0.6**	**0.5**	**0.6**	**0.5**	**0.6**	**0.5**	**0.6**	**0.5**
SHERMAN5	2.8	2.5	2.5	2.4	2.6	2.0	**2.3**	**2.0**	2.4	2.2
	2.9	2.3	2.9	2.3	2.5	2.0	2.7	2.1	2.4	2.0
SWANG1	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**
	0.4	0.3	0.4	0.3	0.4	0.3	**0.3**	**0.3**	0.4	0.3
WANG1	3.2	3.4	**2.9**	**2.6**	3.1	2.7	2.9	2.9	3.0	3.0
	3.2	3.2	3.1	2.8	3.0	2.6	3.0	2.8	3.0	2.8
WATSON4	0.3	0.1	0.4	0.1	**0.2**	**0.0**	0.3	0.1	0.3	0.0
	0.7	0.2	0.5	0.1	0.3	0.0	0.4	0.1	**0.2**	**0.0**
WATSON5	**12.4**	**30.3**	†	†	†	4.8	†	51.9	†	14.7
	†	†	†	†	†	6.0	†	53.7	†	14.1
WATT1	0.3	0.4	0.6	0.3	0.4	**0.3**	**0.3**	0.3	0.6	0.4
	0.3	**0.3**	0.6	0.5	0.6	0.3	0.6	0.2	0.4	0.5
WATT2	3.5	3.1	5.3	3.2	0.4	0.4	0.4	0.5	**0.3**	**0.3**
	2.3	0.3	0.4	0.3	0.6	0.3	**0.3**	**0.3**	**0.3**	**0.3**

REFERENCES

[1] P. Amestoy, T. Davis, and I. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), no. 4, pp. 886-905.

[2] M. Benzi, C. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. of Sci. Comput., 17 (1996) pp. 1135-1149.

[3] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. of Sci. Comput., 19 (1998), no. 3, pp. 968-994.

[4] M. Benzi and M. Tůma, *Numerical experiments with two approximate inverse preconditioners*, BIT, 38 (1998), no. 2, pp. 234-241.

[5] M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*. To appear in SIAM J. of Sci. Comput. (Revised version of Los Alamos National Laboratory Technical Report LA-UR-98-2175, May 1998)

[6] M. Benzi, J. Marin, and M. Tůma, *A two-level parallel preconditioner based on sparse approximate inverses*, in D. Kincaid et. al., eds., Iterative Methods in Scientific Computation II, IMACS Series in Computational and Applied Mathematics, IMACS, NJ, 1999 (in press).

[7] R. Bridson, D. Pierce, and W.-P. Tang, *Refined algorithms for a factored approximate inverse*. In preparation.

[8] E. Chow and Y. Saad, *Approximate inverse techniques for general sparse matrices*, Colorado Conference on Iterative Methods, April 5–9, (1994).

[9] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (3), May 1998.

[10] S. Clift, H. Simon, and W.-P. Tang, *Spectral ordering techniques for incomplete LU preconditioners for CG methods*, manuscript.

[11] S. Clift and W.-P. Tang, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, BIT, 35 (1995), pp. 30–47.

[12] E. D'Azevedo, P. Forsyth, W.-P. Tang, *Towards a cost-effective ILU preconditioner with high level fill*, BIT, 32 (1992), pp. 442–463.

[13] S. Demko, W. Moss, and P. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491-499.

[14] I. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–637.

[15] M. Fiedler, *An algebraic approach to connectivity of graphs*, in *Recent advances in graph theory (Proc. Second Czechoslovak Sympos., Prague, 1974)*. Academia, Prague, 1975, pp. 193-196.

[16] M. Field, *Improving the Performance of Parallel Factorised Sparse Approximate Inverse Preconditioners*, talk presented at the Copper Mountain Conference on Iterative Methods, Colorado, 1998.

[17] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[18] A. George and J. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[19] J. Gilbert, *Predicting structure in sparse matrix computations*, SIAM J. Matrix. Anal. Appl., 15 (1994), pp. 62–79.

[20] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), no. 3, pp. 838–853.

[21] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1999), no. 1, 359-392 (electronic).

[22] L. Kolotilina and A. Yeremin, *Factorized sparse approximate inverse preconditionings I. theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[23] J. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix. Anal. Appl., 11 (1990), pp. 134–172.

[24] W.-P. Tang, *Towards an effective sparse approximate inverse preconditioner*. To appear in SIAM J. Matrix. Anal. Appl., 1998.