SIAM J. SCI. COMPUT.
Vol. 23, No. 5, pp. 1799–1816

MINIMIZING THE PROFILE OF A SYMMETRIC MATRIX*

WILLIAM W. HAGER†

Abstract. New approaches are developed for minimizing the profile of a sparse, symmetric matrix. The heuristic approaches seek to minimize the profile growth, either absolutely or in a weighted sense. The exchange methods make a series of permutations in an initial ordering to strictly improve the profile. Comparisons with the spectral algorithm, a level structure method, and a wave front method are presented.

Key words. matrix envelope, matrix profile, matrix ordering, sparse matrices, Cuthill–McKee algorithm, spectral ordering, wave front ordering, exchange method, weighted greed

AMS subject classifications. 65F05, 65F50

PII. S1064827500379215

1. Introduction.

The envelope of an n by n symmetric matrix $\mathbf{A}$ is the set of index pairs that lie between the first nonzero element in each row and the diagonal:

$$\mathrm{Env}(\mathbf{A}) = \{(i,j) : 1 \le i \le n,\ f_i(\mathbf{A}) \le j < i\},$$

where

$$f_i(\mathbf{A}) = \min\{j : 1 \le j \le i, \ \text{ with } a_{ij} \ne 0\}. \tag{1.1}$$

We assume for convenience that the diagonal elements of $\mathbf{A}$ do not vanish. The profile of a matrix is the number of elements in the envelope plus the number of elements on the diagonal.[1] The $\mathbf{LU}$ factorization of a symmetric, positive definite matrix can be performed within the space associated with the profile.

When solving sparse, symmetric, positive definite linear systems, we often store only the nonzeros. One of the standard storage structures consists of a pointer from the column to arrays holding the row indices and numerical values of the nonzero entries of that column. In contrast, if the matrix is mostly nonzero, the entire n by n matrix is often stored, both zero and nonzero entries. This storage structure eliminates the integer row indices and pointer arrays, but it requires the storage of zero entries as well as nonzeros. Profile storage is useful when the matrix is moderately sparse, or when the nonzero entries are near the main diagonal. In this approach, we only store the location of the first nonzero in each row, along with the numerical values of entries in the row between the first nonzero and the main diagonal. Profile reducing algorithms, like those presented in this paper, try to minimize the profile by making a symmetric permutation of rows and columns.

The graph G associated with $\mathbf{A}$ has vertices

$$\mathcal{V} = \{1, 2, \ldots, n\}$$

and edges

$$\mathcal{E} = \{(i,j) : i \ne j,\ a_{ij} \ne 0\}.$$

*Received by the editors October 9, 2000; accepted for publication (in revised form) May 28, 2001; published electronically February 6, 2002. This work was supported by the National Science Foundation under grant 9704912.
http://www.siam.org/journals/sisc/23-5/37921.html
†Department of Mathematics, University of Florida, Gainesville, FL 32611 (hager@math.ufl.edu, http://www.math.ufl.edu/∼hager).
[1]Some authors do not include the diagonal elements in the profile.

Here the edges are unordered pairs. The first strategies for reducing envelope size were based on graph level structures. A level structure is a partition of the vertices into disjoint sets $L_1, L_2, \ldots, L_k$ with the property that all vertices adjacent to vertices in level L_i are in either L_{i-1} or L_i or L_{i+1}. The first scheme exploiting a level structure to reorder the rows and columns of a matrix and to minimize envelope size was the algorithm of Cuthill and McKee [3]. Later Alan George observed that if the Cuthill–McKee ordering was reversed, then the size of the envelope was often reduced. Liu and Sherman [24] show that this reversal can never increase the size of the envelope. Related level structure methods are developed by Gibbs, Poole, and Stockmeyer [9] and Gibbs and King [23] for which the envelope size is often comparable, but the computation time is significantly less.

Another approach for minimizing the profile is Sloan's method [27], in which the rows and columns of a matrix are ordered in accordance with a priority function; this function assigns to each of the presently unordered nodes a value that is a linear combination of the distance to an "end node" and the associated change in the wave front. In each step of the algorithm, the node with the highest priority is added to the list of labeled nodes. Further improvements to this scheme are developed by Duff, Reid, and Scott [5].

The spectral approach of Barnard, Pothen, and Simon [1] uses an eigenvector (Fiedler vector) corresponding to the second smallest eigenvalue of the graph's Laplacian to minimize the profile of a matrix. The matrix ordering corresponds to the permutation of the components of the Fiedler vector which gives either increasing or decreasing order. George and Pothen [8] provide analysis justifying this strategy. More recently Kumfert and Pothen [22] have developed a hybrid scheme that uses the spectral algorithm to obtain a preordering that is further refined using a modification of Sloan's algorithm. In their numerical experiments, this hybrid algorithm generated profiles 2 to 5 times smaller than those of the reverse Cuthill–McKee algorithm. This was developed further by Reid and Scott [26] and implemented in the HSL [16] package MC60 (HSL was formerly known as the Harwell Subroutine Library).

Recently, Boman and Hendrickson [2] and Hu and Scott [17] developed multilevel algorithms for profile optimization, analogous to the multilevel algorithms that have proved so successful in graph partitioning [13, 14, 18, 19, 20]. Solving the coarse grid problem using an adjacent exchange approach, Boman and Hendrickson achieved performance comparable to that of the original Sloan algorithm. More recently, Hu and Scott achieved performance comparable to that of the hybrid Sloan algorithm by using Sloan's algorithm itself to solve the coarse grid problem. These multilevel methods are faster than the hybrid method since they do not involve computing the Fiedler vector.

In this paper, we develop two classes of methods for optimizing the profile of a matrix. The first class of methods are heuristics based on the following observation: The envelope of a matrix contains all the (off-diagonal) nonzeros of a matrix, and if $\mathbf{P}$ is a permutation matrix, then the number of nonzeros in $\mathbf{A}$ and in $\mathbf{PAP}^\mathsf{T}$ are identical. Hence, finding $\mathbf{P}$ to minimize the profile is equivalent to finding $\mathbf{P}$ to minimize the number of zeros in the envelope. Our algorithm starts in the lower right corner of the matrix and builds up $\mathbf{P}$, minimizing the growth of zeros in the envelope in each step. We consider two variations of this approach: "pure greed," in which the growth of zeros is minimized in each step, and "weighted greed," in which a weighted measure of the growth in zeros is minimized.

Heuristics like these, as well as those mentioned earlier, usually improve the profile; however, improvement is not guaranteed. The other class of methods that we develop involve exchanges of adjacent rows and adjacent columns to strictly improve the profile. Boman and Hendrickson also use adjacent exchanges, along with the techniques of Kernighan and Lin [21] and Fiduccia and Mattheyses [6], to solve their coarse grid problem. Here we consider a series of adjacent exchanges—even though individual exchanges do not improve the profile, a series of exchanges may lead to an improvement. Formulas are derived for the profile change associated with the entire series of exchanges. From the structure of these formulas, we obtain a subset of the possible exchanges that could yield an improvement. These exchange methods can be applied directly to the starting matrix to improve its profile, or they can be applied to the ordering generated by any of the heuristics to further improve the profile. In fact, they could be combined with either of the hybrid methods [22, 26] to obtain a 3-method hybrid, whose profile is at least as good as that of the 2-method hybrid. Numerical comparisons between the ordering strategies are given in section 4 using matrices that arise in linear programming (netlib/LP) and matrices from the Harwell-Boeing and NASA directories of Timothy Davis's University of Florida sparse matrix collection at www.cise.ufl.edu/research/sparse/matrices.

2. The new heuristics. Let $\mathbf{P}$ be a permutation matrix and let $\mathbf{B} = \mathbf{PAP}^{\mathsf{T}}$ be the permutation of $\mathbf{A}$. This permutation $\mathbf{A}$ can be described by a vector $\mathbf{p} \in \mathbf{R}^n$ where p_i is the row of $\mathbf{A}$ corresponding to row i in the permuted matrix $\mathbf{B}$. In our profile heuristic, we assign values to p_i, starting with $i = n$, and working toward $i = 1$. At step k the associated set of "labeled vertices" is defined by

$$\mathcal{L}_k = \{p_i : k < i \leq n\},$$

where the initial set $\mathcal{L}_n$ is the empty set. The complement of $\mathcal{L}_k$, denoted $\mathcal{U}_k$, is the set of "unlabeled vertices." The "wave front" $\mathcal{W}_k$ relative to $\mathcal{L}_k$ and $\mathcal{U}_k$ is defined by

$$\mathcal{W}_k = \{i \in \mathcal{L}_k : a_{ij} \neq 0 \text{ for some } j \in \mathcal{U}_k\}.$$

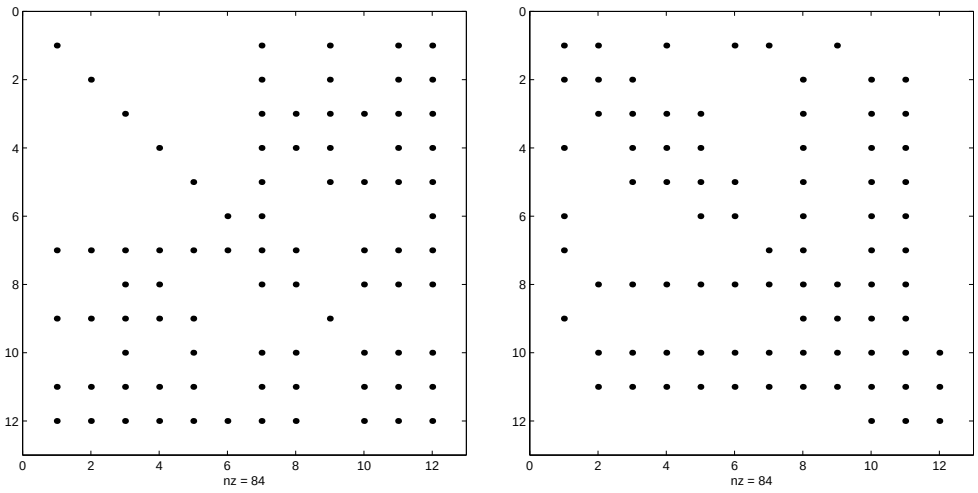
If $\mathbf{A}$ is permuted symmetrically so that the leading columns correspond to $\mathcal{U}_k$ and the trailing columns correspond to $\mathcal{L}_k$, then the wave front is the set of trailing rows where there will be nonzero multipliers in the leading columns when the lower left corner of the permuted matrix is eliminated.

Now let us consider the effect of making the assignment $p_k = j$ at step k for some $j \in \mathcal{U}_k$. The row indices of those elements a_{ij} that both vanish and lie in the envelope of the permuted matrix are given by

$$\mathcal{Z}_k(j) = \{i \in \mathcal{W}_k : a_{ij} = 0\}.$$

As noted earlier, the size of the profile is minimized when the number of zeros in the profile is minimized. A purely greedy labeling approach would simply choose j to minimize the size of $\mathcal{Z}_k(j)$, the number of new zeros that enter the envelope at step k. In other words, if $|\mathcal{Z}_k(j)|$ denotes the number of elements in $\mathcal{Z}_k(j)$, the purely greedy approach chooses $p_k = j \in \mathcal{U}_k$ where

$$|\mathcal{Z}_k(j)| = \min_{l \in \mathcal{U}_k} |\mathcal{Z}_k(l)|. \tag{2.1}$$



A. Original matrix B. Matrix permuted by Algorithm 1

FIG. 1. *Illustration of Algorithm* 1.

A possible choice for the starting node p_n could be a node of minimum degree in the graph of the matrix. Letting d_j denote the degree of node j, this greedy labeling strategy is the following.

ALGORITHM 1 (greedy minimum profile growth).

$J = arg \min_{1 \le j \le n} d_j$
$p_n = j$ *for any* $j \in J$
for $k = n-1, n-2, \ldots, 1$
$J = arg \min_{j \in \mathcal{U}_k} |\mathcal{Z}_k(j)|$
$p_k = j$ *for any* $j \in J$
end
end Algorithm 1

Here the notation "arg min" denotes the set of indices which attain the minimum, while $\mathcal{U}_k$ is the set of unlabeled vertices at step k.

Figure 1 illustrates one of the deficiences with this purely greedy strategy. The last column of Figure 1B corresponds to column 6 in the original matrix and the node of minimum degree. The next four columns (numbers 11 down to 8) in Figure 1B correspond to columns 7, 12, 1, and 11 in Figure 1A. These columns are attractive to the greedy algorithm since they create no zeros in the profile in the lower right 5×5 corner of the matrix. Now, skipping to the end, the first column in Figure 1B corresponds to column 9 in Figure 1A. Notice that when putting column 9 of Figure 1A in column 1 of Figure 1B, we end up with many zeros within the profile (in rows 4, 6, 7, and 9). In this greedy approach, column 9 of Figure 1A was left to the end for the following reason: If we were to assign column 9 earlier in Figure 1B, zeros would be created in the profile in rows 11, 10, and 8. Thus Algorithm 1 does not take into account the fact that these columns are already mostly nonzero, and there is only room for 3 more zeros. As we delay the assignment of column 9 in Figure 1A, we create, in each step, more zeros in the profile in Figure 1B.

To rectify this problem, we can weight our score for each row to take account of the number of nonzeros that remain in a column (or row). The weight function should be chosen so that a zero placed in a column with a lot of nonzeros is less significant than a zero placed in a column with a small number of nonzeros. Let χ be the indicator function defined by

$$\chi(a) = 1 \text{ if } a \neq 0, \quad \chi(a) = 0 \text{ if } a = 0,$$

and for $i \in \mathcal{L}_k$, let $r_k(i)$ be the number of nonzeros associated with unlabeled nodes in row i:

$$r_k(i) = \sum_{j \in \mathcal{U}_k} \chi(a_{ij}). \tag{2.2}$$

In the purely greedy labeling strategy, we assign to each $j \in \mathcal{U}_k$ a "score" $s_k(j)$ that is simply the number of elements in the set $\mathcal{Z}_k(j)$. On the other hand, if W is the weight function, then the weighted score, based on the row sums $r_k(i)$, is

$$s_k(j) = \sum_{i \in \mathcal{Z}_k(j)} W(r_k(i)). \tag{2.3}$$

If W is identically one, then $s_k(j) = |\mathcal{Z}_k(j)|$ and Algorithm 1 corresponds to minimizing $s_k(j)$ in each step. But as illustrated in Figure 1, it is better to choose W so that $W(t)$ is small when t is large. For example, we could take $W(t) = 1/t$. In this way, zeros created in rows with lots of nonzeros receive a small weight in the score for node j. The best node to label at step k is the node $j \in \mathcal{U}_k$ with the lowest score $s_k(j)$.

The formula (2.3) involves a sum over indices associated with zero entries in $\mathbf{A}$. Since sparse matrices have many zeros, it would be better to work with an analogous score expressed in terms of nonzeros. Let $\mathcal{N}_k(j)$ be the nonzeros associated with node j and the wave front at step k:

$$\mathcal{N}_k(j) = \{i \in \mathcal{W}_k : a_{ij} \neq 0\}.$$

The score of node j based on nonzeros is

$$S_k(j) = \sum_{i \in \mathcal{N}_k(j)} W(r_k(i)) = \left(\sum_{i \in \mathcal{W}_k} W(r_k(i)) \right) - s_k(j). \tag{2.4}$$

Again, the weight function W should be monotone decreasing since rows with a lot of nonzeros are less significant than rows with a few nonzeros. Since the scores S_k and s_k are complementary, as indicated in (2.4), minimizing s_k is equivalent to maximizing S_k. Hence, when working with nonzeros, the biggest score is best, and we should choose j to maximize $S_k(j)$ in each step. Again, if W is identically one, then maximizing $S_k(j)$ over $j \in \mathcal{U}_k$ is equivalent to minimizing $s_k(j)$ over $j \in \mathcal{U}_k$, which is the greedy heuristic given in Algorithm 1.

The score $S_k(j)$ takes into account the effect of edges between an unlabeled node j and the labeled nodes $\mathcal{L}_k$. It does not take into account the effect of edges between j and the other unlabeled nodes. If an unlabeled node j is connected by a small number of edges to unlabeled nodes, it should be scored favorably since it is relatively easy to prevent the growth of zeros in the part of the profile connected with this row.

Likewise, as the number of edges between j and the other nodes in $\mathcal{U}_k$ approaches k, this node should be scored favorably since there is little room left for zeros in its row. Our final weighted score $\bar{S}_k(j)$, taking into account both edges between j and $\mathcal{L}_k$ as well as edges between j and $\mathcal{U}_k$, is the following:

$$\bar{S}_k(j) = \max\{W(d_k(j)+1), W(k-d_k(j))\} + \sum_{i\in\mathcal{N}_k(j)} W(r_k(i)), \tag{2.5}$$

where $d_k(j)$ is the degree of j in the subgraph associated with $\mathcal{U}_k$. The first term in (2.5) has the following interpretation: It is the score associated with row k if j is labeled at step k.

The first term in (2.5) leads to a natural starting procedure. For $k = n$, the summation in (2.5) disappears and $\bar{S}_n(j)$ is maximized by either a node of lowest degree or a node of highest degree. Since the best place for completely dense (nonzero) rows and columns is the trailing border of the matrix, this starting procedure treats dense rows and columns correctly. For completeness, we state the weighted greed scheme.

ALGORITHM 2 (weighted greed).

for $k = n, n-1, \ldots, 1$
  $J = \arg\max_{j\in\mathcal{U}_k} \bar{S}_k(j)$
  $p_k = j$ *for any* $j \in J$
end

end Algorithm 2

If the weight function is strictly monotone, then evaluation of scores could be computationally expensive, with a work estimate comparable to that of an **LU** factorization itself. By choosing W to be piecewise constant, the scoring process is sped up since the score for any node will change only when one of its associated $r_k(i)$, $i \in \mathcal{N}(j)$, moves across a step in W. In the numerical experiments reported later, we used a weight function of the following form:

$$W(i) = 1/i \text{ for } 1 \le i \le \psi, \quad W(i) = 1/\psi \text{ for } i > \psi, \tag{2.6}$$

where ψ is a small positive integer which we call the cutoff. In the experiments, $\psi = 8$. For the matrix of Figure 1A, the profile generated by Algorithm 2 is 54, while the profile of Figure 1B and Algorithm 1 is 63. The notation nnz, used below, stands for "number of nonzeros."

LEMMA 2.1. *For the weight function* (2.6)*, Algorithm* 2 *can be implemented in time bounded by* $\mathrm{O}(\mathrm{nnz}(\mathbf{A})\log n)$.

Proof. In each step of Algorithm 2, $\mathbf{r}_k$, the vector whose ith component is $r_k(i)$, is adjusted according to the location of nonzeros in column p_k of $\mathbf{A}$. Hence, the number of components of $\mathbf{r}_k$ and $\mathbf{r}_{k-1}$ that are different is bounded by the number of nonzeros in column p_k of $\mathbf{A}$. It follows that the total number of changes in components of $\mathbf{r}_k$ as k ranges between n and 1 is bounded by $\mathrm{nnz}(\mathbf{A})$. If $W(r_k(i)) \ne W(r_{k-1}(i))$ for some i, then $r_k(i) \le \psi$ according to (2.6). For each $i \in \mathcal{L}_k$, $r_k(i)$ can only decrease as k decreases since the set $\mathcal{U}_k$ in (2.2) decreases in size as k decreases from n down to 1. Hence, for any given i and for the weight function (2.6), there are at most ψ values of k for which $W(r_k(i))$ changes. Now consider the last term in (2.5), which we need to evaluate for each $j \in \mathcal{U}_k$. We can think of this sum in the following way: For each $i \in \mathcal{L}_k$ associated with a nonzero a_{ij}, $j \in \mathcal{U}_k$, let us replace a_{ij} by $W(r_k(i))$, and let $\mathbf{A}_k$ be the resulting matrix whose remaining elements are all zero. Forming the

sum in (2.5) for each $j \in \mathcal{U}_k$ is equivalent to computing the column sums of $\mathbf{A}_k$; the only nonzero entries are in a submatrix in the lower left corner of $\mathbf{A}_k$. As k decreases by one, we delete a column and add a row to the submatrix. If the numbers inside the submatrix were fixed, independent of k, then the work associated with adding a row, deleting a column, and updating the column sums is bounded by the number of nonzeros in $\mathbf{A}$. Note though that not only is the submatrix changing (add a row, delete a column) in each step, but the elements inside the submatrix change too. However, for each choice of i, we saw earlier that there are at most ψ values of k for which $W(r_k(i))$ changes its value. Hence, the total number of changes in value of the elements in the submatrix is bounded by $\psi\text{nnz}(\mathbf{A})$, and the total work in maintaining the sum in (2.5) remains $\text{O}(\text{nnz}(\mathbf{A}))$.

In each step of Algorithm 2, $\mathbf{d}_k$, the vector whose entries are the degrees of the nodes in the subgraph associated with $\mathcal{U}_k$, is adjusted according to the location of nonzeros in column p_{k+1} of $\mathbf{A}$: For each $i \in \mathcal{U}_k$, we have

$$d_k(i) = d_{k+1}(i) - 1$$

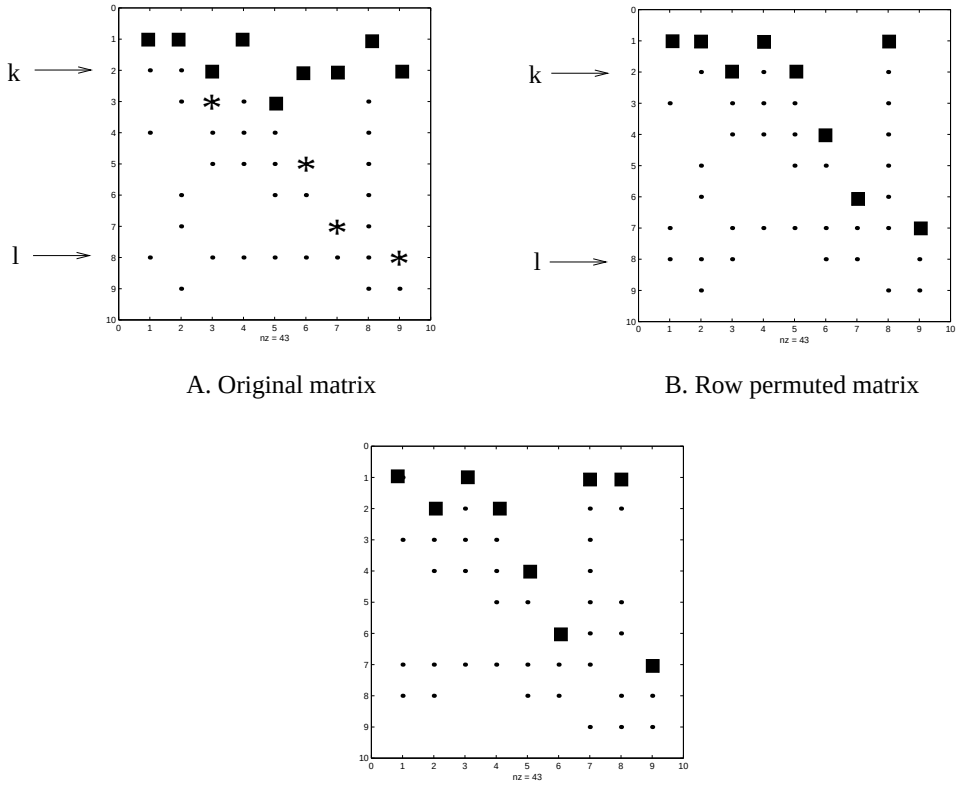
if $a_{ij} \neq 0$ for $j = p_{k+1}$. As with $\mathbf{r}_k$, the total number of changes in components of $\mathbf{d}_k$ as k ranges between n and 1 is bounded by $\text{nnz}(\mathbf{A})$. In any step k where $d_k(j) < d_{k+1}(j)$ for some $j \in \mathcal{U}_k$, we need to check whether the max in (2.5) has increased. Since the total number of changes in components of $\mathbf{d}_k$ is bounded by $\text{nnz}(\mathbf{A})$, the time needed to check whether the max in (2.5) increases when $d_k(j)$ decreases is $\text{O}(\text{nnz}(\mathbf{A}))$.

Now consider the second term in the max of (2.5). Assume that each $j \in \mathcal{U}_k$ is stored in a list, ordered by degree. This degree-ordered list can be created and maintained in time bounded by $\text{O}(\text{nnz}(\mathbf{A}))$ since the number of changes in degree is bounded by $\text{O}(\text{nnz}(\mathbf{A}))$. Since $k - d_k(j)$ decreases monotonically as k decreases, there are at most ψ values of k where $W(k - d_k(j))$ changes in value. Hence, the total number of pairs (j, k) for which $W(k - d_k(j))$ changes in value is at most ψn. Since the diagonal of $\mathbf{A}$ does not vanish, $\psi n = \text{O}(\text{nnz}(\mathbf{A}))$. Consequently, the effort involved with checking whether the max in (2.5) increases due to an increase in $W(k - d_k(j))$ is $\text{O}(\text{nnz}(\mathbf{A}))$.

In each step of Algorithm 2, we extract an unlabeled node of minimum score. If the nodes are ordered in a heap according to their score, we can update the heap when a node's score changes in time bounded by $\text{O}(\log k) \leq \text{O}(\log n)$. Since the number of terms that change in the scores (2.5) is at most $\text{O}(\text{nnz}(\mathbf{A}))$, the total work in maintaining the heap is bounded by the product $\text{O}(\text{nnz}(\mathbf{A}) \log n)$. □

3. Exchange methods. By an exchange method, we mean any strategy for improving the profile of a matrix based on a series of symmetric interchanges of rows and of columns. One strategy to improve the profile of a matrix would be to consider all possible pairs of rows, and corresponding columns, and make an exchange if the profile is reduced. In this section, however, we focus on more specialized exchanges.

For any n by n matrix $\mathbf{A}$ and $k < l \leq n$, let $D_{k:l}(\mathbf{A})$ denote the change in the profile associated with a series of adjacent exchanges: Interchange row k with $k+1$, row $k+1$ with $k+2$, ..., row $l-1$ with l, and perform the symmetric interchange of columns. Before stating a formula for $D_{k:l}(\mathbf{A})$, we give an illustration. Consider the 9 by 9 matrix depicted in Figure 2, and the values $k = 2$ and $l = 8$. Throughout this section, our figures are based on the "skyline view" of matrix profile—in other words, elements in each column between the first nonzero and the diagonal. The dots denote nonzero entries and a solid square is placed over each nonzero entry corresponding to a frontier node, a node on the top boundary of the envelope. The first nonzero beneath



A. Original matrix

B. Row permuted matrix

C. Row and column permuted matrix

FIG. 2. *Downward exchanges in a symmetric matrix.*

a frontier node is covered with an asterisk. When row k is exchanged with row $k+1$, row $k+1$ is exchanged with $k+2$, and so on, down to row l, we obtain Figure 2B. The single frontier node beneath row k is moved one level higher, increasing the profile by 1. The frontier nodes in row k, on the other hand, fall to positions right above the asterisks in Figure 2A, reducing the profile by 11. Altogether, the profile is reduced by 10. To complete the permutation, we perform the corresponding exchange of the columns. This second set of permutations restores symmetry and moves the columns around, but it does not affect the height of frontier nodes—there are still 4 frontier nodes in row 1, 2 in row 2, and so on. Since the number of zeros above the frontier nodes does not change, the profile change deduced in Figure 2B is correct.

Our formula for $D_{k:l}$ is the following.

LEMMA 3.1. *For any $k < l \le n$ and for f defined in* (1.1)*, we have*

$$D_{k:l}(\mathbf{A}) = |\{j \ge k : k < f_j(\mathbf{A}) \le l\}| - \left(\sum_{j \in \mathcal{F}_k} (\min\{l, g_j(\mathbf{A}) - 1\} - k) \right), \tag{3.1}$$

where

$$\mathcal{F}_k = \{j : f_j(\mathbf{A}) = k\}, \tag{3.2}$$

and

$$g_i(\mathbf{A}) = \min\{j : f_i(\mathbf{A}) < j,\ a_{ij} \neq 0\} \tag{3.3}$$

if the minimum exits. Otherwise, $g_i(\mathbf{A}) = n + 1$.

The elements of the frontier set $\mathcal{F}_k$, the dark squares in Figure 2, are nodes that are adjacent to k and on the boundary of the envelope. The value of g_i is the second nonzero column index in row i, if it exists. The elements of $\mathbf{A}$ corresponding to the g_i and $i \in \mathcal{F}_k$ are the asterisks in Figure 2. Note that $g_i(\mathbf{A})$ could be larger than i, while $f_i(\mathbf{A}) \leq i$.

Proof. The first set in (3.1) represents those zeros that are added to the profile as we successively exchange the rows between k and l. The summation in (3.1) corresponds to the zeros removed from the profile as the rows are exchanged and the frontier moves towards the diagonal of the matrix. Once we reach the second nonzero in a row, the frontier in that row remains stationary. To complete the permutation, we perform the corresponding exchange of the columns. This second set of permutations restores symmetry and moves the columns around, but it does not affect the number of zeros above each frontier node. Since the profile of the matrix equals the number of elements in the upper triangle minus the number of zeros above the frontier nodes, this second set of permutations does not affect the profile (since the number of zeros above the frontier nodes does not change). □

Suppose that we wish to determine whether a series of downward exchanges can be performed that would reduce the profile of a symmetric matrix. Referring to (3.1), we see that as l increases, the first term increases while the second term decreases, achieving its minimum value when l reaches the maximum of the $(g_j(\mathbf{A}) - 1)$ over $j \in \mathcal{F}_k$. When $l \geq g_j(\mathbf{A}) - 1$ for all $j \in \mathcal{F}_k$, the second term in (3.1) remains constant. Hence, when determining the best value for l, we should restrict our attention to

$$k < l < l_1 := \max_{j \in \mathcal{F}_k} g_j(\mathbf{A}).$$

Also note that for $l > k$, we have the bound

$$\sum_{j \in \mathcal{F}_k} (\min\{l, g_j(\mathbf{A}) - 1\} - k) \leq \sum_{j \in \mathcal{F}_k} (g_j(\mathbf{A}) - k - 1).$$

The first term in (3.1) is a monotone increasing function of l, so when minimizing $D_{k:l}(\mathbf{A})$ over $l > k$, we should require that $l \leq l_2$ where l_2 is the largest l with the property that

$$|\{j \geq k : k < f_j(\mathbf{A}) \leq l\}| \leq m_1 := \sum_{j \in \mathcal{F}_k} (g_j(\mathbf{A}) - k - 1).$$

Lemma 3.1 can be utilized in an exchange scheme to improve a profile in the following way: For each $k = n-1, n-2, \ldots, 1$, we could check whether $D_{k:l}(\mathbf{A}) < 0$ for some $l > k$. If L is a value l that minimizes $D_{k:l}(\mathbf{A})$ over $l > k$, and if $D_{k:L}(\mathbf{A}) < 0$, we should exchange the rows between k and L. In searching for a value of l that minimizes $D_{k:l}(\mathbf{A})$, the constraints $l \leq l_1$ and $l \leq l_2$ should be exploited to reduce the search space. In the following summary of the downward exchange scheme, we let $\mathbf{P}$ denote the permutation matrix associated with the permutation vector $\mathbf{p}$.

ALGORITHM 3 (down exchanges).

for $k = n-1, n-2, \ldots, 1$
  $L = arg \min_{k<l\le n} D_{k:l}(\mathbf{PAP}^{\mathsf{T}})$
  if $D_{k:l}(\mathbf{A}) < 0$ *for some* $l \in L$
    $q = p_k; \quad p_{k:l-1} = p_{k+1:l}; \quad p_l = q;$
  end if
end

end Algorithm 3

Algorithm 3 can be performed in concert with Algorithm 2. That is, right after making the assignment $p_k = j$ in Algorithm 2, we could evaluate the L of Algorithm 2 and make a downward exchange if it is beneficial. This combined mode works since downward exchanges in the trailing part of the matrix do not change the scores of unlabeled nodes. This combined mode avoids a second pass over the matrix. For the experiments reported in section 4, we implement this combined mode; however, the improvement in the profile connected with the down exchanges is accumulated and subtracted from the total profile. This allows us to run the code in this more efficient coupled mode, while tabulating separately the profile improvement due to the exchanges.

Similar to the downward exchanges, we could perform upward exchanges to improve the profile. For any matrix $\mathbf{A}$ and $k > l$, let $D_{k:l}(\mathbf{A})$ denote the change in the profile associated with the interchange of rows k and $k-1$, rows $k-1$ and $k-2$, ..., rows $l-1$ and l, and the symmetric interchange of columns. The effect of upward exchanges is quite different from that of downward exchanges. As an illustration, consider the 7 by 7 symmetric matrix in Figure 3. Again, a solid black square is placed over each nonzero corresponding to a frontier node. When row 7 is exchanged successively with the rows above it, the frontier node in column 7 is moved up to row 1. This increases the profile by 3. At the same time, the frontier nodes in columns 1, 2, 3, and 6, which lie above zeros in row 7, are lowered by one row. The net decrease in the profile is 1. When the corresponding column interchanges are applied to the matrix of Figure 3B, symmetry is restored. Since these column exchanges do not affect the number of frontier nodes in each row, the change in profile deduced in Figure 3B is correct.

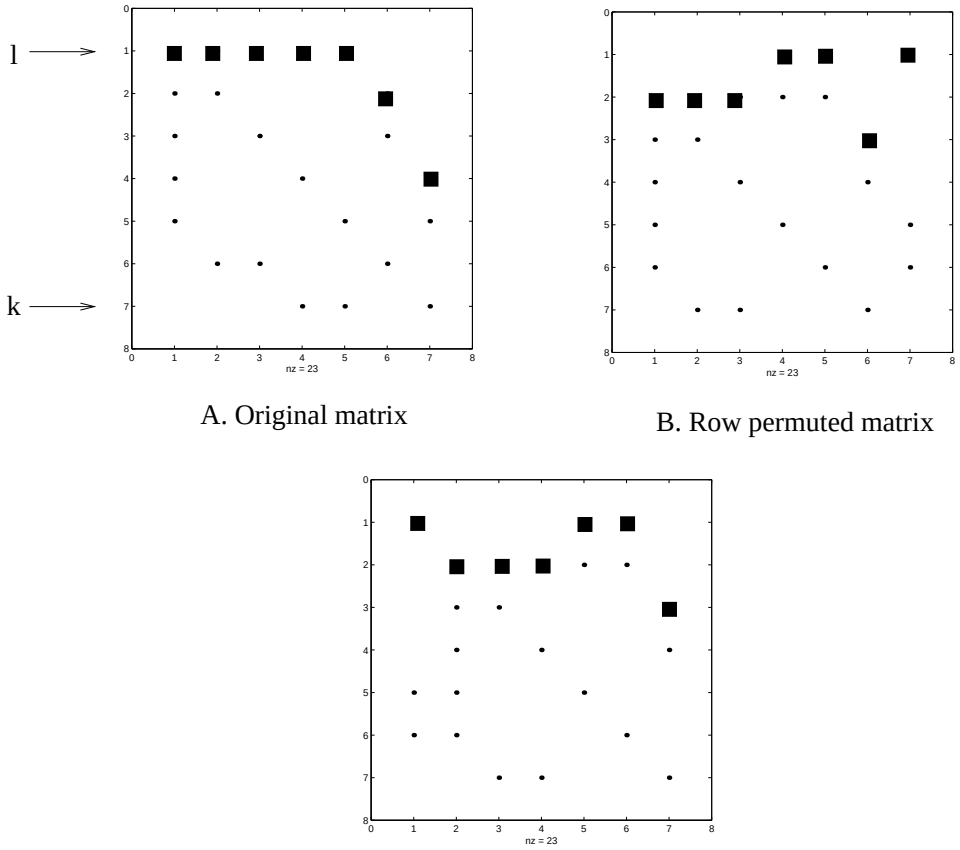
LEMMA 3.2. *For any* $l < k \le n$ *and for* f *defined in* (1.1)*, we have*

$$D_{k:l}(\mathbf{A}) = \left(\sum_{\{j:a_{kj}\neq 0\}} \max\{f_j(\mathbf{A}) - l, 0\} \right) - |\{j : a_{kj} = 0,\ k > f_j(\mathbf{A}) \ge l\}|. \tag{3.4}$$

Proof. As row k is exchanged successively with rows above, the zeros in row k cause the frontier nodes above them to drop one row lower. The second term in (3.4) reflects this effect. On the other hand, as row k is exchanged upward, nonzeros in row k eventually reach the frontier, and each subsequent exchange increases the number of zeros in the profile by one. The first term in (3.4) reflects the effect of these nonzeros in row k. To complete the permutation, we perform the corresponding exchange of the columns. These column exchanges restore symmetry but do not affect the number of zeros above each frontier node. Hence, this second set of permutations does not affect the profile. ∎

Let l_3 be the largest value of $l < k$ for which

$$\sum_{a_{kj}\neq 0} \max\{f_j(\mathbf{A}) - l, 0\} \ge m_2 := |\{j : k > f_j(\mathbf{A}) \ge 1\}|.$$



A. Original matrix

B. Row permuted matrix

C. Row and column permuted matrix

FIG. 3. *Upward exchanges in a symmetric matrix.*

Since the first term in (3.4) increases as l decreases, and since the second term in (3.4) is no bigger than m_2, we conclude that $D_{k:l}(\mathbf{A}) \geq 0$ for all $l \leq l_3$. Hence, when searching for the minimum of $D_{k:l}(\mathbf{A})$ over $l < k$, we should restrict our attention to $l > l_3$.

By (3.4) $D_{k:l}(\mathbf{A})$ is a piecewise linear function of l whose minimum is attained either at $l = k$ or $l = f_j(\mathbf{A})$ for some j. Consequently, to achieve rapid evaluation of the minimum of $D_{k:l}(\mathbf{A})$, we should record for each value of m, the sizes of the level sets

$$\mathcal{F}_m = \{j : f_j(\mathbf{A}) = m\},$$

the same set appearing in Lemma 3.1, and those m for which $\mathcal{F}_m$ are nonempty should be ordered in a linked list. The minimum of $D_{k:l}(\mathbf{A})$ is achieved, either at $l = k$ or at one of the m in this list.

Lemma 3.2 can be utilized in an exchange scheme in much the same way that we utilize Lemma 3.1 in Algorithm 3. However, instead of starting at the bottom of the matrix and exchanging with rows below, we start at the top of the matrix and exchange with rows above. In the following summary of the upward exchange scheme, we again let $\mathbf{P}$ denote the permutation matrix associated with the permutation vector $\mathbf{p}$.

ALGORITHM 4 (up exchanges).
for $k = 2, 3, \ldots, n$
$L = arg\ \min_{1 \le l < k}\ D_{k:l}(\mathbf{PAP}^{\mathsf{T}})$
if $D_{k:l}(\mathbf{A}) < 0$ *for some* $l \in L$
$q = p_l; \quad p_{l:k-1} = p_{l+1:k}; \quad p_k = q;$
end if
end
end Algorithm 4

The down exchange and up exchange schemes are utilized in a iterative fashion; we apply Algorithm 3, followed by Algorithm 4, followed by Algorithm 3, and so on. In practice, there was no significant improvement after a relatively small number of sweeps, where a sweep denotes a pass through Algorithm 3 followed by a pass through Algorithm 4. In many cases, there was no improvement at all after 3 sweeps, while in some cases, there was a small improvement even after 20 sweeps. In the numerical experiments, we limited the number of sweeps to 5. The execution time of exchange methods can be reduced, while limiting the possible profile improvement, by restricting the extent of the adjacent rows that are exchanged. In our experiments, we did not allow exchanges that went beyond 1000 adjacent row swaps in each exchange step. Another approach to speed up the exchange process, presented in [25], is to use a bisection step to determine the rows to exchange.

4. Numerical experiments. In this section, we report on a series of numerical experiments using three test sets, all found in Timothy Davis's University of Florida sparse matrix collection, currently located at

www.cise.ufl.edu/research/sparse/matrices

In particular, we use all 109 matrices in the linear programming test set (most of David Gay's linear programming set found originally in Netlib), the 153 symmetric, nondiagonal matrices in the Harwell-Boeing collection [4], and the 8 symmetric matrices in the NASA test set (matrices which arise in structural engineering problems).

The linear programming test set includes m by n matrices $\mathbf{A}$ where m is between 24 and 105,127, with an average value of 3236. The first series of experiments were based on a set of 14,842 matrices, which we call Test Set 1, constructed in the following way: Using the Chaco partitioning code of Hendrickson and Rothberg [15], we generated a permutation matrix $\mathbf{P}$ designed so that $\mathbf{PA}(\mathbf{PA})^{\mathsf{T}}$ has as many nonzeros as possible in diagonal blocks of size roughly k by k where $k = 2\text{nnz}(\mathbf{A})/m$. This construction arises when one uses block iterative techniques to solve linear programming problems (see [10, 11, 12]). The diagonal blocks generated in this way were used for the test matrices. These matrices have an average density (fraction of nonzeros) .45, while their average dimension is 13.1. If the matrices are simply stored as dense symmetric matrices (only store the upper triangle), the storage requirement would be 3,400,729. Using suitable permutations and profile storage, the storage is reduced by a factor of more than 2.

Test Set 2 is obtained in the following way: For each of the 109 rectangular matrices, we form $\mathbf{AA}^{\mathsf{T}}$. A matrix with this nonzero pattern arises in interior point methods as well as in the LP dual active set algorithm. The average density of these matrices is .11, four times smaller than the density of the matrices in Test Set 1. The matrices in Test Set 3, the Harwell-Boeing symmetric matrices, have an average dimension of 2043 and an average density of .03, about four times smaller than the average density of Test Set 2. The matrices in Test Set 4, the NASA symmetric matrices, have an average dimension of 15,750, and an average density of .007, about

TABLE 1
Profiles, units of 10^6*, for Test Set* 1.

w	wx	m	mx	r	rx	s	sx	x
1.299	1.278	1.303	1.288	1.583	1.313	1.983	1.292	1.272

four times smaller than the average density of Test Set 3. The names of the matrices in Test Sets 2, 3, and 4 and the nonzero patterns for the matrices in Test Set 1 can be found at

www.math.ufl.edu/∼hager/papers/profile_testsets

We consider the following five methods:

1. Algorithm 2 (weighted greed).
2. Sloan's method as implemented in Algorithm MC60 from HSL [16] (provided by John Reid).
3. Reverse Cuthill–McKee algorithm (routine symrcm of Matlab).
4. The spectral method (using Matlab's eig or eigs routines).
5. Algorithms 3 and 4 (exchange methods).

MC60 allows for several algorithmic modes; supernodes can be exploited, reverse Cuthill–McKee ordering can be done besides Sloan's method, a spectral method can be combined with Sloan's method by specifying a priority function, and an interface to a frontal solver is provided. In our experiments, we use simply Sloan's method without supernodes (none of the codes in our experiments exploit supernodes). In the fifth method above, we iterate Algorithms 3 and 4 until either there is no further improvement in the profile or until we have done 5 iterations. We initially apply Algorithms 3 and 4 to the given matrix; then we generate a random permutation of the matrix and repeat Algorithms 3 and 4. We report the best profile gotten after 10 random permutations of the given matrix. Since Algorithms 3 and 4 can be used to improve the profile of any given matrix, we also applied them to the permuted matrix generated by each of the first four algorithms. In our experiments, Algorithm 2 nearly always provided a better profile than Algorithm 1; consequently, results for Algorithm 1 are not given.

In Table 1, we give the total profile for the 14,842 matrices in the first test set. The column headings are the following:

w – weighted greed, Algorithm 2,
wx – weighted greed followed by Algorithms 3 and 4, the exchange methods,
m – MC60,
mx – MC60 followed by the exchange methods,
r – reverse Cuthill–McKee,
rx – reverse Cuthill–McKee followed by the exchange method,
s – spectral method,
sx – spectral method followed by the exchange methods,
x – apply the exchange methods to the original matrix and 10 permutations of it.

Observe that the smallest profile was gotten by method x, the exchange methods combined with 10 random permutations. The second best profile was gotten by method wx, the weighted greed/exchange method combination. All the methods received significant benefit from postprocessing using the exchange methods. The biggest beneficiary was the spectral method, for which the profile improved about 35%.

TABLE 2
Unique superiority for Test Set 1.

wx	mx	rx	sx	x
45	27	19	80	227

TABLE 3
Profiles for Test Set 2.

Dim	Scale	#	w	wx	m	mx	r	rx	s	sx	x
100	10^3	11	7.1	7.0	7.4	7.2	8.6	7.6	8.8	7.6	6.9
400	10^5	29	2.2	2.1	2.3	2.1	3.4	2.3	2.9	2.1	2.1
1600	10^6	40	2.7	2.5	3.0	2.7	4.7	3.1	2.7	2.5	2.7
6400	10^7	19	2.9	2.8	3.2	3.1	4.4	3.3	4.2	3.0	3.2
25,600	10^7	7	3.0	2.7	4.6	4.2	14.1	7.7	10.0	5.5	5.9
>25,600	10^8	3	.6	.6	9.2	8.7	15.8	9.9	11.2	8.8	.9

Although all methods generated the same profile for many of the 14,842 matrices in the first test set, there were some cases where one method was uniquely superior. As expected, we see in Table 2 that the exchange method x was more often uniquely superior than the other methods. Surprisingly, the spectral/exchange combination sx, ranking fourth in Table 1, was uniquely superior in 80 cases. Although this is a small number of cases relative to the 14,842 problems, it is significantly larger than the number of cases for any of the other methods in Table 2.

In order to estimate the deviation between the profiles of Table 1 and the smallest possible profile, we considered 10,000 random permutations of each matrix. In each case, we applied Algorithm 3 and 4 and recorded the smallest profile. For these relatively small matrices, the smallest profile generated from these 10,000 starting guesses should be very close to the smallest possible profile. In fact, over the entire set of 14,842 matrices, there were only 14 cases where any of the heuristics generated a profile strictly better than the best obtained from these 10,000 random starting guesses, and in these 14 cases, the heuristics yielded a total improvement of just 137. Hence, we expect that the smallest profile for these 14,842 matrices is very close to the profile 1,263,882 achieved using the 10,000 random starting guesses. Referring to Table 1, method x differs from the estimated optimal profile by less than a percent.

In Test Set 2, we apply each of the profile schemes to the matrix $\mathbf{A}\mathbf{A}^\mathsf{T}$ associated with each linear program. In Table 3, the first column indicates the dimension range, the row scale factor, and the number of matrices. For example, the first row of Table 3 corresponds to a set of 13 matrices whose dimensions lie between 1 and 100, and the profiles in that row should be multiplied by 10^3 to obtain the total profile of the 13 matrices in that class. The second row corresponds to a set of 29 matrices of dimension between 101 and 400, and the profiles in that row should be multiplied by 10^5. For the matrices of dimension up to 400, the exchange method x gave the best results. For larger matrices, weighted greed combined with the exchange methods gave the best profiles. As the dimension of a matrix increases, the 10 random permutations used in method x become an increasingly small subset of the total set of permutations.

Running times on a Sun Ultra 10 computer are given in Table 4 for weighted greed, MC60, and the exchange methods. The time for the exchange method is subdivided into x3, corresponding to Algorithm 3, and x4, corresponding to Algorithm 4. The

TABLE 4
Total running times for Test Set 2.

Dim	w	wx	m	mx	x3	x4
100	.060	.120	.003	.011	.006	.009
400	.728	1.527	.083	.241	.090	.271
1600	3.222	7.748	.569	2.297	.750	3.603
6400	6.183	23.600	5.530	16.680	4.328	21.264
25,600	9.064	63.460	1.392	48.560	19.136	66.509
>25,600	18.430	157.800	2.684	186.900	648.636	258.818

times for x3 and x4 were the total times of Algorithms 3 and 4 divided by 11 (the number of permutations including the starting matrix). MC60 is coded in Fortran while the other algorithms are coded in C. Note that MC60 runs several times faster than weighted greed. The complexity estimate $O(\text{nnz}(\mathbf{A})+n\log n)$ for Sloan's method, given in [22], when compared to the complexity estimate $O(\text{nnz}(\mathbf{A})\log n)$ for weighted greed in Lemma 2.1, also seems to suggest that Sloan's method should be faster than our current implementation, which is patterned after the proof of Lemma 2.1. Observe that when the exchange methods are applied after either weighted greed or Sloan's method, overall run time can increase by several factors. Nonetheless, the time to factor a matrix is often much less than the time used by the exchange method. For example, Matlab's built-in Cholesky factorization routine (chol) applied to a matrix with the same nonzero pattern as that of the eighth NASA problem, skirt, takes 33 secs, compared to .1 secs by MC60 to generate a low profile permutation, and 3.7 secs for both MC60 and the exchange methods. The combination mx reduces the profile from 3,766,168 down to 687,730.

Focusing on the times for the exchange methods in Table 4, we note that Algorithm 3 is generally much faster than Algorithm 4. The one exception is the last line of Table 4. This line corresponds to 3 matrices, and one of these, ken18, dominates the total time. This matrix, with 105,127 rows, comes from the manufacturer with a relatively nice row ordering, and a random permutation generates a horrendous ordering. Algorithm 3, which is the first one to process the permuted matrix, makes many interchanges while improving the profile. Even though Algorithm 3 is faster than Algorithm 4, the number of interchanges is so large that the Algorithm 3 time dominates.

We emphasize that the x3 and x4 times in Table 4 correspond to the exchange methods applied to a single permuted version of the original matrix (the time was evaluated by averaging over a set of permutations). It is clear from the times given in Table 4 that trying to find an optimal profile by randomly permuting the matrix and improving the profile by exchanges is only appropriate when the matrix is relatively small.

In Table 5 we give the profiles associated with the Harwell-Boeing test set. These matrices are generally more structured than the linear programming matrices, and more sparse. Again, the exchange methods were competitive for dimensions up to 400. Thereafter, the MC60/exchange combination mx or the spectral/exchange combination sx gave the best profiles.

Running times, shown in Table 6, again parallel those of Table 4, with MC60 being the fastest code.

The profiles shown in Table 7 for Test Set 4, the largest and sparsest matrices, were similar to those of Table 5, with either mx or sx giving the best profiles in each case.

TABLE 5
Profiles for Test Set 3.

Dim	Scale	#	w	wx	m	mx	r	rx	s	sx	x
100	10^4	19	1.43	1.42	1.32	1.30	1.49	1.32	1.40	1.29	1.34
400	10^5	31	1.14	1.12	1.05	1.02	1.23	1.05	1.20	1.02	1.08
1600	10^6	71	2.07	2.00	1.82	1.73	2.36	1.86	2.21	1.88	2.56
6400	10^6	24	8.28	8.08	5.66	5.55	7.04	6.26	6.16	5.53	9.45
25,600	10^7	5	4.23	3.98	1.62	1.57	2.36	2.16	2.59	2.00	1.89
>25,600	10^7	3	8.58	8.52	7.28	7.27	9.91	9.84	5.12	4.54	13.60

TABLE 6
Total running times for Test Set 3.

Dim	w	wx	m	mx	x3	x4
100	.121	.247	.006	.021	.012	.026
400	.645	1.328	.051	.161	.053	.245
1600	6.375	15.330	.412	3.808	.927	9.236
6400	6.906	23.260	.717	10.360	2.696	22.191
25,600	7.040	25.300	1.139	15.280	4.222	35.100
>25,600	13.420	41.390	3.324	17.400	6.366	78.282

TABLE 7
Profiles for Test Set 4.

Dim	Scale	#	w	wx	m	mx	r	rx	s	sx	x
6400	10^6	4	2.28	2.26	1.64	1.64	1.89	1.86	1.73	1.48	1.79
25,600	10^6	2	2.84	2.82	1.34	1.29	1.81	1.67	1.79	1.61	11.23
>25,600	10^7	2	3.15	3.14	2.41	2.38	2.63	2.55	2.52	2.35	8.18

TABLE 8
Total running times for Test Set 4.

Dim	w	wx	m	mx	x3	x4
6400	1.249	2.545	.207	.451	.191	2.143
25,600	2.414	10.950	.199	8.347	2.216	17.627
>25,600	10.700	51.580	1.887	42.670	18.964	81.400

Again, in Table 8, we see that MC60 is significantly faster than the other codes.

These experiments seem to indicate that for matrices in the test suite with dimensions up to 400, the best or close to the best profiles are obtained by the exchange methods applied to a small number of random permutations of the starting matrix. If the exchange methods are applied after MC60, the profile improves as it must, but the computing time increases by several factors (at least in our implementation). By restricting the distance between the rows and columns that are exchanged, the execution time of the exchange methods is $O(\mathrm{nnz}(\mathbf{A}))$. For the matrices $\mathbf{A}\mathbf{A}^T$ associated with the linear programming test programs, weighted greed combined with the exchange methods provided the best profiles on average for matrices of dimension greater than 400. For the Harwell-Boeing and NASA test sets, which are generally more structured than the linear programming matrices, Sloan's method was the best

single method, obtaining the best profile for each class of matrices except for the 3 matrices of dimension greater than 25,600 in Table 5. When the profiles for the single methods were further improved using the exchange methods, the combination sx achieved a better profile than mx in 5 of the 9 matrix classes associated with Tables 5 and 7.

Acknowledgment. The author gratefully acknowledges the thoughtful and detailed comments of the referees, which led to an improved presentation.

REFERENCES

[1] S. T. Barnard, A. Pothen, and H. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, Numer. Linear Algebra Appl., 2 (1995), pp. 317–334.

[2] E. G. Boman and B. Hendrickson, *A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices*, Tech. Report SCCM-96-14, Stanford University, Stanford, CA, 1996.

[3] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the ACM National Conference, ACM, New York, 1969, pp. 157–172.

[4] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[5] I. S. Duff, J. K. Reid, and J. A. Scott, *The use of profile reduction algorithms with a frontal code*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2555–2568.

[6] C. M. Fiduccia and R. M. Mattheyses, *A linear time heuristic for improving network partitions*, in Proceedings of the 19th IEEE Design Automation Conference, Las Vegas, NV, 1982, IEEE, Piscataway, NJ, 1982, pp. 175–181.

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

[8] A. George and A. Pothen, *An analysis of spectral envelope reduction via quadratic assignment problems*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 706–732.

[9] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM J. Numer. Anal., 13 (1976), pp. 236–250.

[10] W. W. Hager, *Iterative methods for nearly singular linear systems*, SIAM J. Sci. Comput., 22 (2000), pp. 747–766.

[11] W. W. Hager, *The LP dual active algorithm*, in High Performance Algorithms and Software in Nonlinear Optimization, R. De Leone, A. Murli, P. M. Pardalos, and G. Toraldo, eds., Kluwer, Dordrecht, 1998, pp. 243–254.

[12] W. W. Hager, C.-L. Shih, and E. O. Lundin, *Active Set Strategies and the LP Dual Active Set Algorithm*, Department of Mathematics, University of Florida, Gainesville, FL, 1996; available from www.math.ufl.edu/∼hager.

[13] B. Hendrickson and R. Leland, *A Multilevel Algorithm for Partitioning Graphs*, Tech. Report SAND93-1301, Sandia National Laboratories, Albuquerque, NM, 1993.

[14] B. Hendrickson and R. Leland, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM J. Sci. Comput., 16 (1995), pp. 452–469.

[15] B. Hendrickson and E. Rothberg, *Improving the run time and quality of nested dissection ordering*, SIAM J. Sci. Comput., 20 (1998), pp. 468–489.

[16] HSL, *A Collection of Fortran Codes for Large Scale Scientific Computation*, 2000; available online from http://www.cse.clrc.ac.uk/Activity/HSL.

[17] Y. F. Hu and J. A. Scott, *Multilevel Algorithms for Wavefront Reduction*, RAL-TR-2000-031, Rutherford Appleton Laboratory, Chilton, England, 2000; available online from www.numerical.rl.ac.uk/reports/reports.html.

[18] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[19] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, J. Parallel Distrib. Comput., 48 (1998), pp. 96–129.

[20] G. Karypis and V. Kumar, *Parallel multilevel k-way partitioning scheme for irregular graphs*, SIAM Rev., 41 (1999), pp. 278–300.

[21] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Tech. J., 29 (1970), pp. 291–307.

[22] G. Kumfert and A. Pothen, *Two improved algorithms for reducing the envelope and wavefront*, BIT, 37 (1997), pp. 1–32.

[23] J. G. Lewis, *Implementations of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms*,

ACM Trans. Math. Software, 8 (1982), pp. 180–189.
[24] W.-H. Liu and A. H. Sherman, *Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices*, SIAM J. Numer. Anal., 13 (1976), pp. 198–213.
[25] J. K. Reid and J. A. Scott, *Implementing Hager's exchange methods for matrix profile reduction*, RAL-TR-2001-039, Rutherford Appleton Laboratory, Chilton, England, 2001.
[26] J. K. Reid and J. A. Scott, *Ordering symmetric sparse matrices for small profile and wavefront*, Internat. J. Numer. Methods Engrg., 45 (1999), pp. 1737–1755.
[27] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Internat. J. Numer. Methods Engrg., 23 (1986), pp. 239–251.