SIAM J. Sci. Comput.
Vol. 32, No. 6, pp. 3426–3446
© 2010 Society for Industrial and Applied Mathematics

HYPERGRAPH-BASED UNSYMMETRIC NESTED DISSECTION ORDERING FOR SPARSE LU FACTORIZATION*

LAURA GRIGORI[†], ERIK G. BOMAN[‡], SIMPLICE DONFACK[†], AND TIMOTHY A. DAVIS[§]

Abstract. In this paper we discuss a hypergraph-based unsymmetric nested dissection (HUND) ordering for reducing the fill-in incurred during Gaussian elimination. It has several important properties. It takes a global perspective of the entire matrix, as opposed to local heuristics. It takes into account the asymmetry of the input matrix by using a hypergraph to represent its structure. It is suitable for performing Gaussian elimination in parallel, with partial pivoting. This is possible because the row permutations performed due to partial pivoting do not destroy the column separators identified by the nested dissection approach. The hypergraph nested dissection approach is essentially equivalent to graph nested dissection on the matrix A^TA, but we need only the original matrix A and never form the usually denser matrix A^TA. The usage of hypergraphs in our approach is fairly standard, and HUND can be implemented by calling an existing hypergraph partitioner that uses recursive bisection. Our implementation uses local reordering constrained column approximate minimum degree (CCOLAMD) to further improve the ordering. We also explain how weighted matching (HSL routine MC64) can be used in this context. Experimental results on 27 medium and large size matrices with highly unsymmetric structures compare our approach to four other well-known reordering algorithms. The results show that it provides a robust reordering algorithm, in the sense that it is the best or close to the best (often within 10%) of all the other methods, in particular on matrices with highly unsymmetric structures.

Key words. sparse LU factorization, reordering techniques, hypergraph partitioning, nested dissection

AMS subject classifications. 65F50, 65F05, 68R10

DOI. 10.1137/080720395

1. Introduction.

Solving a sparse linear system of equations $Ax = b$ is an important operation for many scientific applications. Gaussian elimination can be used to accurately solve these systems and consists of decomposing the matrix A into the product of L and U, where L is a lower triangular matrix and U is an upper triangular matrix. Due to the operations associated with the Gaussian elimination, some of the elements that are zero in the original matrix become nonzero in the factors L and U. These elements are referred to as fill elements. Hence one of the important preprocessing steps preceding the numerical computation of the factors L and U consists in reordering the equations and variables such that the number of fill elements is reduced.

Although this problem is NP-complete [39], in practice there are several efficient fill-reducing heuristics. They can be divided into two groups. The first group uses

*Received by the editors April 7, 2008; accepted for publication (in revised form) September 3, 2010; published electronically November 30, 2010.
http://www.siam.org/journals/sisc/32-6/72039.html

[†]INRIA Saclay - Ile de France, Laboratoire de Recherche en Informatique Universite Paris-Sud 11, France (laura.grigori@inria.fr, sidonfack@gmail.com). The first and third author's work was supported by an X-Scale-NL grant from DIGITEO and Région Ile-de-France.

[‡]Scalable Algorithms Dept., Sandia National Laboratories, Albuquerque, NM 87185-1318. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. This author's work was supported by the United States Department of Energy Office of Science through the CSCAPES SciDAC Institute (egboman@sandia.gov).

[§]CISE Dept., University of Florida, Gainesville, FL 32611. This author's work was supported by the National Science Foundation (0620286) (davis@cise.ufl.edu).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

greedy heuristics to locally reduce the number of fill elements at each step of Gaussian elimination. One of the representative heuristics is the minimum degree algorithm. This algorithm uses the graph associated with a symmetric matrix to permute it such that the vertex of minimun degree is eliminated at each step of the factorization. Several variants, such as multiple minimum degree [34] (Mmd) and approximate minimum degree [1] (Amd), improve the minimum degree algorithm in terms of time and/or memory usage.

The second group is based on global heuristics and uses graph partitioning to reorder the matrix such that it has fill only in some specific blocks. Nested dissection [20] is the main technique used in the graph partitioning approach. This algorithm uses the graph of a symmetric matrix and employs a divide-and-conquer approach. The graph partitioning approach has the advantage of reordering the matrix into a form suitable for parallel execution. State-of-the-art nested dissection algorithms use multilevel partitioning [26, 32]. A widely used routine is Metis_NodeND from the Metis [31] graph partitioning package.

It has been observed in practice that minimum degree is better at reducing the fill for smaller problems, while nested dissection works better for larger problems. This observation has lead to the development of hybrid heuristics that consist of applying several steps of nested dissection, followed by the usage of a local greedy heuristic on local blocks [27].

For unsymmetric matrices, the above algorithms use the graph associated with the symmetrized matrix $A+A^T$ or A^TA. The approach of symmetrizing the input matrix works sometimes well. However, computing the nonzero structure of A^TA can be time consuming in practice. There are few heuristics that compute an ordering for A^TA without explicitly forming the structure of A^TA. The column approximate minimum degree [12] (Colamd) is a local greedy heuristic that implements the approximate minimum degree algorithm on A^TA without computing the structure of A^TA. The approach of wide separators [6] is a global approach that orders the graph of A^TA by widening separators obtained in the graph of $A^T + A$. In [6], the wide separator method was combined with Colamd local ordering. We use a similar hybrid approach, but we do not use wide separators.

There are few approaches in the literature that aim at developing fill-reducing algorithms that use as input the structure of A alone. For local heuristics, this is due to the fact that the techniques for improving the runtime of minimum degree are difficult to extend to unsymmetric matrices. In fact the minimum degree algorithm was developed from the Markowitz algorithm [37], which was proposed earlier for unsymmetric matrices. In the Markowitz algorithm, the degree of a vertex (called the Markowitz count) is defined as the product of the number of nonzeros (nnzs) in the row and the nnzs in the column corresponding to this vertex. It is expensive to determine the Markowitz count at each step of the algorithm. Hence, a recent local greedy heuristic for unsymmetric matrices [2] uses local symmetrization techniques to approximate the Markowitz count associated with each vertex. This work has been extended in [3] where the numerical values of the input matrix are also taken into account.

In this paper we present one of the first fill-reducing ordering algorithms that fully exploits the asymmetry of the matrix and that is also suitable for parallel execution. We refer to this algorithm as the hypergraph-based unsymmetric nested dissection (Hund). It uses a variation of nested dissection, but it takes into account the asymmetry of the input matrix by employing a hypergraph to represent its nonzero structure. Using this hypergraph, the algorithm first computes a hyperedge separator of the entire hypergraph that divides it into two disjoint parts. The matrix is reordered

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

such that the columns corresponding to the hyperedge separator are ordered after those in the disconnected parts. The nested dissection is then recursively applied to the hypergraph of the two disconnected parts, respectively.

The usage of hypergraphs in our algorithm is fairly standard. The algorithm can be implemented by calling one of the existing hypergraph partitioners that uses recursive bisection. However, there are two main contributions in this paper. The algorithm uses as input the structure of A alone and does not need to compute the structure of A^TA. We discuss later in the paper the relations between using the hypergraph of A or the graph of A^TA to reorder A. In addition we show how the partitioning algorithm can be used in combination with other important techniques in sparse Gaussian elimination. This includes permuting large entries on the diagonal [17] and a technique for improving stability in solvers implementing Gaussian elimination with static pivoting [35] as well as other local orderings.

An important property of our approach is that the structure of the partitioned matrix is insensitive to row permutations. In other words, the row permutations induced by pivoting during Gaussian elimination do not destroy the column separators. Hence the fill is reduced because it can occur only in the column separators and in the disconnected parts of the matrix. But also this property is particularly important for a parallel execution, since the communication pattern, which depends on the column separators, can be computed prior to the numerical factorization. In addition, the partitioning algorithm can be used in combination with other important techniques in sparse Gaussian elimination.

We note that a partitioning algorithm that takes into account the asymmetry of the input matrix was also considered by Duff and Scott in [18, 19]. There are several important differences with our work. The authors focus on the parallel execution of LU factorization, and their goal is to permute the matrix to a so-called singly bordered block diagonal (SBBD) form. In this form the matrix has several diagonal blocks (which can be rectangular), and the connections between them are put together in the columns ordered at the end of the matrix. The advantage of this form is that the diagonal blocks can be factorized independently, though special care must be taken since the blocks are often nonsquare. The authors rely on a different approach for computing this form, and they do not analyze it for fill-reducing ordering.

The remainder of the paper is organized as follows. In section 2 we give several basic graph theory definitions, and we describe in detail the nested dissection process. In section 3 we present our Hund algorithm and its different steps. Section 4 discusses relations to graph models as the graph of A^TA or the bipartite graph of A. In section 5 we present experimental results that study the effectiveness of the new algorithm, in terms of fill, on a set of matrices with highly unsymmetric structures. We also compare the results with other fill-reducing ordering algorithms. Finally, section 6 concludes the paper.

2. Background: Nested dissection and hypergraphs.

2.1. Nested dissection.

Nested dissection [20, 33] is a fill-reducing ordering method based on the divide-and-conquer principle. The standard method applies only to symmetric matrices; here we show a nonsymmetric variation.

The sparsity structure of a structurally symmetric matrix is often represented as an undirected graph. The nested dissection method is based on finding a small vertex separator, S, that partitions the graph into two disconnected subgraphs. If we order the rows and columns corresponding to the separator vertices S last, the permuted

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

matrix PAP^T has the form

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{13}^T & A_{23}^T & A_{33} \end{pmatrix},$$

where the diagonal blocks are square and symmetric. Now the diagonal blocks A_{11} and A_{22} can be factored independently and will not cause any fill in the zero blocks. We propose a similar approach in the nonsymmetric case, based on a column separator. Suppose we can permute A into the form

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix},$$

where none of the blocks are necessarily square. (This is known as SBBD form.) Then we can perform Gaussian elimination, and there will be no fill in the zero blocks. Furthermore, this property holds even if we allow partial pivoting and row interchanges. Note that if any of the diagonal blocks are square, then A is reducible and the linear systems decouple.

The question is how to obtain an SBBD structure with a small column separator. There are two common approaches: a direct approach [5, 29], and indirect methods that first find doubly bordered block diagonal form [19]. We choose the direct method and use hypergraph partitioning.

2.2. Hypergraph partitioning and ordering. In this paper we consider the hypergraph model to represent the nonzero structure of an unsymmetric matrix. Other graph models are briefly discussed in section 4. A hypergraph $H(V,E)$ contains a set of vertices V and a set of hyperedges E (also known as *nets*), where each hyperedge is a subset of V. We will use the column-net hypergraph model of a sparse matrix [7] where each row corresponds to a vertex and each column is a hyperedge. Hyperedge e_j contains the vertices given by the nonzero entries in column j. An example of a matrix A is given in (2.1), and its hypergraph is shown in Figure 2.1.

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & & & & & & & & x \\ & x & & & & x & & & \\ & x & & & x & & & x & \\ x & & x & x & & & & & \\ & & & & x & x & & x & \\ & & & & & x & & & x \\ & & x & & & & x & & \\ x & & & x & & & & & \\ & & & x & & & x & x & \end{pmatrix} \end{array}. \tag{2.1}$$

Suppose we partition the vertices (rows) into two sets, R_1 and R_2. This induces a partition of the hyperedges (columns) into three sets: C_1, C_2, and C_3. Let C_1 be the set of hyperedges (columns) where all the vertices (rows) are in R_1. Similarly, let C_2 be the set of hyperedges (columns) where all the vertices (rows) are in R_2. Let C_3 be the set of hyperedges (columns) that are "cut"; that is, they have some vertices in R_1 and some in R_2. Note that some of the sets R_i or C_j could be empty, though this rarely happens in practice. Now let P be a row permutation such that all of R_1 is

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

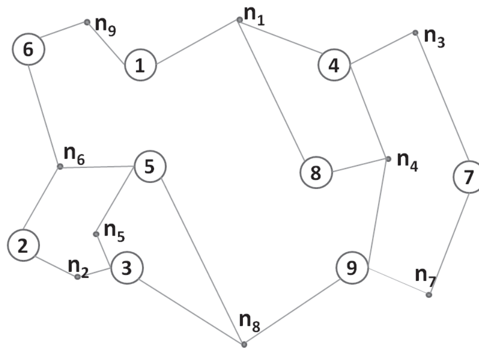

FIG. 2.1. *Hypergraph of matrix A in* (2.1). *The large circles are vertices, and the small black dots represent hyperedges (nets).*

ordered before R_2, and let Q be a column permutation such that all of C_1 is ordered before C_2, and all of C_2 is ordered before C_3. Then

$$PAQ = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \end{pmatrix}. \tag{2.2}$$

If one of the sets C_j is empty, then the corresponding column block in (2.2) is also empty, and the decomposition simplifies. Even when C_1 and C_2 are nonempty, it may happen that some rows in A_{11} or A_{22} are all zero. In this case, permute such rows to the bottom, and we get

$$\bar{P}AQ = \begin{pmatrix} \bar{A}_{11} & 0 & \bar{A}_{13} \\ 0 & \bar{A}_{22} & \bar{A}_{23} \\ 0 & 0 & \bar{A}_{33} \end{pmatrix}. \tag{2.3}$$

Observe that when A has full rank, blocks A_{11} ($\bar{A}_{11}$) and A_{22} ($\bar{A}_{22}$) cannot have more columns than rows.

Figure 2.2 displays the result of the first step of unsymmetric nested dissection applied on the hypergraph in Figure 2.1. The matrix obtained after permuting matrix A in (2.1) is presented in (2.4). This equation cannot be further reduced to the form presented in (2.3).

$$PAQ = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left(\begin{array}{ccccccccc} & x & & & & & & x & \\ x & & & x & & & & & \\ & x & & x & & & & & \\ x & & x & & & & & & x \\ & & x & x & & & & & x \\ & & & & x & & x & x & \\ & & & & x & x & & & \\ & & & & & & x & x & \\ & & & & & x & x & & x \end{array} \right) \end{array}. \tag{2.4}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

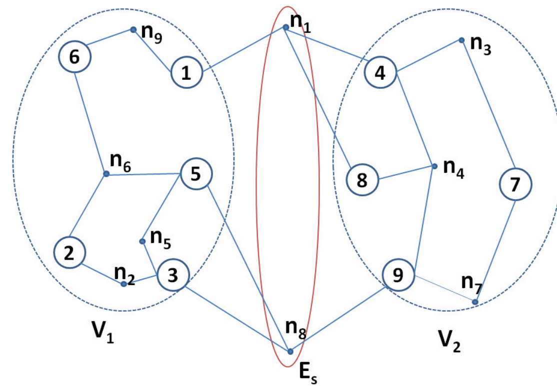

FIG. 2.2. *The result of the first step of unsymmetric nested dissection applied on the hypergraph in Figure* 2.1*; the hyperedge separator is* $E_s = \{n_1, n_8\}$.

Hypergraph partitioning has been well studied [7]. The objective is to partition the vertices into two parts such that there are approximately equally many vertices in each part, and the number of cut hyperedges is minimized. Although it is an NP-hard problem, fast multilevel algorithms give good solutions in practice. Good software is readily available, like PaToH [8], hMetis [30], and Zoltan [16]. The k-way partitioning problem ($k > 2$) is usually solved by recursive bisection.

3. An algorithm using unsymmetric nested dissection. We present an algorithm that uses unsymmetric nested dissection and has three stages. First, we apply HUND to limit the fill. Second, we perform row interchanges based on numerical values to reduce pivoting. Third, we apply a local reordering on local blocks to again reduce fill, following the state-of-the-art hybrid approach used in ordering methods. We note that the second stage is optional, and it is not always used by existing solvers. We include it here to show how it can be used in combination with the unsymmetric nested dissection ordering. We refer to the first step as HUND and to the overall process as *enhanced* HUND.

3.1. Hypergraph recursive ordering. Recall our goal is to find permutations P_1 and Q_1 such that $(P_1AQ_1)(Q_1^Tx) = P_1b$ is easier to solve than the original system $Ax = b$. Our idea is to apply the block decomposition (2.2) recursively. This is quite different from most recursive approaches for ordering linear systems because our blocks are usually not square.

Figure 3.1(left) shows the sparsity structure of P_1AQ_1 after two levels of bisection. We continue the recursive bisection until each block is smaller than a chosen threshold. As in symmetric nested dissection, we expect it is beneficial to switch to a local ordering algorithm on small blocks, but in principle one could continue the recursion until each block has only one row or column. We sketch the recursive ordering heuristic in Algorithm 1. In this variant, the recursion stops at a constant block size, $t_{\min}$. Alternatively, we can partition the hypergraph into a fixed number of parts, k, and derive the column separators from the partitioning. An advantage of the latter approach is we can use standard hypergraph partitioning tools and only call the partitioner once.

Algorithm 1 differs from the more common SBBD method (cf. Duff and Scott [18, 19]) in that we use the SBBD technique recursively. It differs from traditional

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

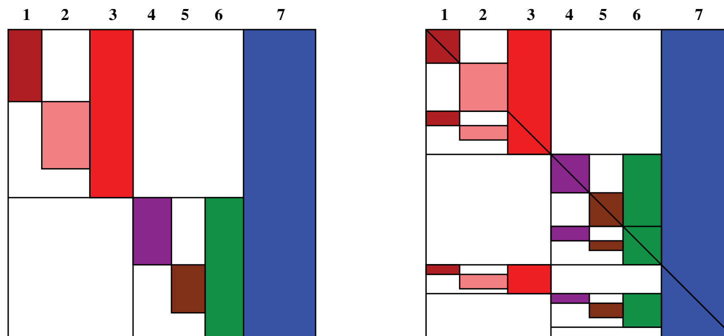

Fig. 3.1. *The matrix after hypergraph ordering (left) and after row permutation from matching (right).*

(symmetric) nested dissection method because the row and column permutations usually differ and were derived by hypergraph partitioning instead of vertex separators. Although recursive hypergraph bisection has been described in several papers, Algorithm 1 is presented here in the context of sparse matrix ordering.

Algorithm 1 HUND (Hypergraph Unsymmetric Nested Dissection).

1: Function $[p, q] = \text{Hund}(A)$
2: $[m, n] = \text{size}(A)$
3: **if** $\min(m, n) \leq t_{\min}$ **then**
4: p= 1:m
5: q= 1:n
6: **else**
7: Create column-net hypergraph H for A
8: Partition H into two parts using hypergraph partitioning
9: Let p and q be the row and column permutations, respectively, to permute A into the block structure in (2.2)
10: $[m_1, n_1] = \text{size}(A_{11})$
11: $[p_1, q_1] = \text{Hund}\ (A_{11})$
12: $[p_2, q_2] = \text{Hund}\ (A_{22})$
13: $p = p(p_1, p_2 + m_1)$
14: $q = q(q_1, q_2 + n_1)$
15: **end if**

3.2. Stabilization. The ordering procedure above takes only the structure into account and not the numerical values. An approach that can be used to stabilize the factorization or to minimize pivoting is to permute large entries to the diagonal. A standard approach is to model this as matching in the bipartite graph [17], and the HSL [28] routine MC64 can be used. This permutation step for obtaining a strong diagonal can be helpful for dynamic (partial) pivoting methods, since the number of row swaps can be significantly reduced, thereby speeding up the factorization process [17]. It is essential for static pivoting methods [35] because it decreases the probability of encountering small pivots during the factorization. However, this step is optional, and it is not always used by existing solvers. We do not use it in the experimental results in section 5. We discuss it here such that we can explain how it can be performed in combination with Hund.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

The matrix obtained after applying the matching permutation to permute the rows on the matrix at the left in Figure 3.1 is shown at the right in Figure 3.1. Observe that after row permutation, the diagonal blocks are now square. The remaining rows in the originally rectangular blocks have been "pushed down". All the permutations applied on the matrix after this step should be symmetric. Note, however, that the following steps do not need the diagonal blocks to be square.

3.3. Local reordering. The goal of this step is to use local strategies to further decrease the fill in the blocks of the permuted matrix. Algorithms such as constrained AMD (CAMD) [11] or constrained COLAMD (CCOLAMD) [11] can be used for this step. These algorithms are based on AMD, respectively, COLAMD, and have the property of preserving the partitioning obtained by the unsymmetric nested dissection algorithm. This is because in a constrained ordering method, each node belongs to one of up to n constraint sets. In our case, a constraint set corresponds to a separator or a partition. After the ordering it is ensured that all the nodes in set zero are ordered first, followed by all the nodes in set one, and so on.

A preprocessing step useful for the efficiency of direct methods consists of reordering the matrix according to a postorder traversal of its elimination tree. This reordering tends to group together columns with the same nonzero structure, so they can be treated as a dense matrix during the numeric factorization. This allows for the use of dense matrix kernels during numerical factorization and hence leads to a more efficient numeric factorization.

In order to preserve the structure obtained in the previous steps, we compute the elimination tree corresponding to the diagonal blocks of the input matrix. Note that in practice, postordering a matrix preserves its structure but can change the fill in the factors L and U. We remark that the local reordering should be applied symmetrically so that the diagonal is preserved.

3.4. Algorithm summary. In summary, the LU factorization with partial pivoting based on enhanced HUND contains several distinct steps in the solution process (step 2 is optional):

1. Reorder the equations and variables by using the HUND heuristic that chooses permutation matrices P_1 and Q_1 so that the number of fill-in elements in the factors L and U of P_1AQ_1 is reduced.
2. Choose a permutation matrix P_2 so that the matrix $P_2P_1AQ_1$ has large entries on the diagonal. The above permutation helps for some solvers to ensure the accuracy of the computed solution.
3. Find a permutation matrix P_3 using a local heuristic and a postorder of the elimination tree associated with the diagonal blocks such that the fill-in is further reduced in the matrix $P_3P_2P_1AQ_1P_3^T$. In our tests this is achieved using constrained COLAMD and postordering based on the row merge tree [22] of the diagonal blocks.
4. Compute the numerical values of L and U.

The execution of the above algorithm on a real matrix (FD18) is displayed in Figure 3.2. The structure of the original matrix is presented at top left. The structure obtained after the unsymmetric nested dissection HUND is presented at top right. The structure obtained after permuting to place large entries on the diagonal using MC64 is displayed at bottom left. Finally, the structure obtained after the local ordering is displayed at bottom right.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

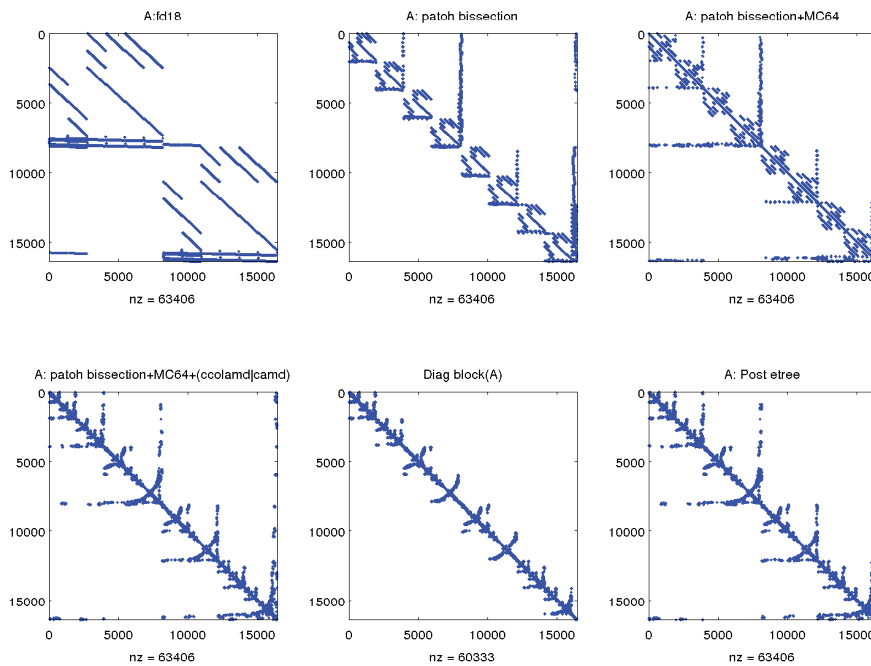

FIG. 3.2. *Example of application of preprocessing steps on a real matrix* FD18*. Displayed are* (a) *the structure of the original matrix* FD18*,* (b) *the structure obtained after* HUND*,* (c) *after MC*64*, and* (d) *after* CCOLAMD.

4. Relation to graph models. We have chosen to work with the hypergraph of A (column-net model). Alternatively, we could have worked with a graph derived from A, such as the graph of A^TA or the bipartite graph of A. Another option is to use the wide separator approach discussed in [6]. In this section we discuss relations between the hypergraph model and these three other approaches.

The graph corresponding to the structure of $B = A^TA$ is known as the column intersection graph, since $B_{ij} \neq 0$ iff columns i and j of A intersect. Catalyürek [10] and Catalyürek and Aykanat [9] showed that a hyperedge separator for A is a vertex separator in B, and proposed to use hypergraph separators in A to obtain a fill-reducing ordering for B. Their main application was linear systems from interior-point methods for linear programming. In addition we show here that a vertex separator in B also gives a hyperedge separator in A; thus these problems are essentially equivalent.

THEOREM 4.1. *A column (hyperedge) separator for A is a vertex separator for A^TA and vice versa.*

Proof. ($\Rightarrow$): Let I_S be the index set of a column (hyperedge) separator in A, and let I_0, I_1 be the index sets for the two remaining parts. (Note one of I_0 or I_1 could be empty.) Therefore, columns in I_i intersect only columns in I_i and I_S for $i = 0, 1$. In the column intersection graph (A^TA), let V_i correspond to columns I_i and V_S correspond to I_S. It follows that there are no edges between V_0 and V_1; therefore V_S is a vertex separator. ($\Leftarrow$): Let V be the vertices in the graph of A^TA, let V_S be a separator, and let V_i, $i = 0, 1$, be the remaining vertex sets. By construction, there is no edge between V_0 and V_1. An edge in $G(A^TA)$ corresponds to a path of length two in the bipartite graph or hypergraph of A. Thus the rows corresponding to V_0 and V_1 are only connected through the columns (hyperedges) corresponding to

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

V_S. When these are removed, the rows corresponding to V_0 and V_1 are no longer connected; hence V_S is a column (hyperedge) separator in A. $\square$

Another approach consists of using the bipartite graph of A. The bipartite graph of the $n \times n$ matrix A is equivalent to the graph of the $2n \times 2n$ matrix

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}.$$

The separators obtained from bisection partitioning of the graph of C can be used to permute the matrix A to a doubly bordered block diagonal form. This approach is used, for example, in [18].

It is easy to see that a vertex separator in the graph of C might not be a column separator in the hypergraph of A. Consider the simple matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} x & & \\ & x & \\ x & x & x \end{pmatrix} \end{array}.$$

Its augmented matrix C has the following structure:

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} & & & x & & \\ & & & & x & \\ & & & x & x & x \\ x & & x & & & \\ & x & x & & & \\ & & x & & & \end{pmatrix} \end{array}.$$

The graph of the augmented matrix C can be partitioned in two parts $\{1, 4\}$ and $\{2, 5, 6\}$, and the vertex separator is formed by $\{3\}$. The permuted matrix C becomes

$$PCP^T = \begin{array}{c} \\ 1 \\ 4 \\ 2 \\ 5 \\ 6 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 4 & 2 & 5 & 6 & 3 \end{array} \\ \begin{pmatrix} & x & & & & \\ x & & & & & x \\ & & & x & & \\ & & x & & & x \\ & & & & & x \\ & x & & x & x & \end{pmatrix} \end{array}.$$

The vertex 3 corresponds to the last row of A, which has nonzeros in all the columns of the matrix A. Hence it is not possible to partition the column-net hypergraph of A in two disconnected parts as in (2.2).

In other words, vertices in the separator of the graph of C which correspond to row vertices of A might not belong to the column separator of the hypergraph of A. Note that when the separator of the graph of C consists purely of column vertices, then these do in fact form a column (hyperedge) separator in the hypergraph of A. Instead of using hypergraphs, we could have used bipartite graphs, but the hypergraph model is more natural since we can use hypergraph partitioning (not a constrained version of bipartite graph partitioning).

The approach presented in [6] consists of finding a wide separator in $G(A^T + A)$ which corresponds to a separator in $G(A^T A)$. The wide separator in $G(A^T + A)$ is defined as being formed by the vertices of a vertex separator of $G(A^T + A)$ plus their

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

adjacent vertices in one of the disconnected subgraphs. The authors show that a wide separator in $G(A^T+A)$ is a separator in $G(A^TA)$. However, the converse is not always true. The authors show that the converse is true for matrices that are structurally symmetric and that have a nonzero diagonal.

Hence in general a separator in the hypergraph $H(A)$, which corresponds to a separator in the graph $G(A^TA)$, will not be a wide separator in the graph $G(A^T+A)$. Consider the example presented in [6]:

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} x & & x \\ & x & x \\ & & x \end{pmatrix} \end{array}.$$

Vertex 3 is a separator in $G(A^T + A)$. A wide separator is formed by vertices $\{1, 3\}$ or by vertices $\{2, 3\}$.

Note that column 3 of A is a column/hyperedge separator in $H(A)$ as well as a separator in $G(A^TA)$. But it does not correspond to a wide separator in $G(A^T + A)$. Hence the wide separator approach is different and may produce larger separators than the approach we propose in this paper.

5. Experimental results. In this section we present experimental results for enhanced Hund ordering applied to real world matrices. The tests are performed on a 2.4 GHz dual core Intel, with 3.3 GB of memory. As stated in the introduction, our goal is to reorder the matrix into a form that is suitable for parallel computation, while reducing or at least maintaining comparable the number of fill-in elements in the factors L and U to other state-of-the-art reordering algorithms. In the experiments, we compare the performance of the new ordering algorithm with other widely used ordering algorithms such as Amd, Colamd, and Metis (nested dissection) in terms of size of the factors. We study the execution time of the new ordering compared to other reordering algorithms. We also discuss the quality of the partitioning in terms of size of the separators.

We use a set of matrices with highly unsymmetric structures that represent a variety of application domains. We present in Table 5.1 their characteristics, which include the matrix order, the nnzs in the input matrix A, the nonzero pattern symmetry, the numerical symmetry, and the application domain. The nonzero pattern symmetry represents the percentage of nonzeros for which both a_{ij} and a_{ji} are nonzeros. The numerical symmetry is computed as the percentage of nonzeros for which $a_{ij} = a_{ji}$. The matrices are available from the University of Florida sparse matrix collection [14] and are listed according to their nnzs. Some of the matrices in our test set are highly reducible such as lhr71c, mult_dcop_03, raefsky6. These matrices can benefit from a different preprocessing in which they are first permuted to a block upper triangular form using the Dulmage–Mendelsohn decomposition and then an ordering heuristic is applied on each diagonal block. However, we keep these matrices in our test set since several solvers such as UMFPACK [13], SuperLU [15], and SuperLU_DIST [35] do not apply the Dulmage–Mendelsohn decomposition and can benefit from our approach.

We compare the hypergraph-based ordering with four widely used fill-reducing ordering algorithms, that is, Amd, Colamd, and Metis nested dissection (applied on the structure of $A + A^T$ or on the structure of A^TA). The quality of each algorithm can be evaluated using several criteria, as the nnz entries in the factors L and U, the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Table 5.1
Benchmark matrices and their characteristics: the order, the nnzs nnz(A), the nonzero pattern symmetry (Str. Sym.), the numeric symmetry (Num. Sym.), and the application domain.

#	Matrix	Order n	$nnz(A)$	Str. sym.	Num. sym.	Application domain
1	SWANG1	3169	20841	100%	0%	Semiconductor device sim
2	LNS_3937	3937	25407	85%	0%	Fluid dynamics
3	POLI_LARGE	15575	33074	0%	0%	Chemical process simulation
4	MARK3JAC020	9129	56175	7%	1%	Economic model
5	FD18	16428	63406	0%	0%	Crack problem
6	LHR04	4101	82682	2%	0%	Light hydrocarbon recovery
7	RAEFSKY6	3402	137845	0%	0%	Incompressible flow
8	SHERMANACB	18510	145149	15%	3%	Circuit simulation
9	BAYER04	20545	159082	0%	0%	Chemical process simulation
10	ZHAO2	33861	166453	92%	0%	Electromagnetism
11	MULT_DCOP_03	25187	193216	61%	1%	Circuit simulation
12	JAN99JAC120SC	41374	260202	0%	0%	Economic model
13	BAYER01	57735	277774	0%	0%	Chemical process simulation
14	SINC12	7500	294986	2%	0%	Single material crack problem
15	ONETONE1	36057	341088	7%	4%	Circuit simulation
16	MARK3JAC140SC	64089	399735	7%	1%	Economic model
17	AF23560	23560	484256	100%	0%	Airfoil eigenvalue calculation
18	E40R0100	17281	553562	31%	0%	Fluid dynamics
19	SINC15	11532	568526	1%	0%	Single material crack problem
20	ZD_JAC2_DB	22835	676439	30%	0%	Chemical process simulation
21	LHR34C	35152	764014	0%	0%	Light hydrocarbon recovery
22	SINC18	16428	973826	1%	0%	Single material crack problem
23	TORSO2	115967	1033473	99%	0%	Bioengineering
24	TWOTONE	120750	1224224	24%	11%	Circuit simulation
25	LHR71C	70304	1528092	0%	0%	Light hydrocarbon recovery
26	AV41092	41092	1683902	0%	0%	Unstructured finite element
27	BBMAT	38744	1771722	53%	0%	Computational fluid dynamics

number of floating point operations performed during the numerical factorization, and the factorization time. We restrict our attention to the first criterion, the nnzs in the factors L and U. (We also computed the number of operations, and the results were quite similar to the nonzero counts.)

The Hund reordering heuristic presented in Algorithm 1 starts with the hypergraph of the input matrix and partitions it recursively into two parts. The recursion is stopped when a predefined number of parts is reached. In our tests we use PaToH [8] (with a fixed seed of 42 for the random number generator) to partition the hypergraph in a predefined number of parts. PaToh uses recursive multilevel bisection, and hence it implements hypergraph partitioning as described in Algorithm 1 and allows us to permute the matrix to the form presented in (2.3). In PaToH, we set *cuttype* = $PATOH_CUTPART$ to minimize the number of hyperedges cut, which corresponds to the number of columns in the border block. We had initially implemented Hund calling PaToH for each bisection step as in Algorithm 1, but this lead to slow execution times, due to excessive copying of the submatrices. Here we report only on the version where we partition into a predefined number of parts, and the recursive bisection is handled efficiently within PaToH.

To study the performance of Hund we vary the number of parts (denoted as k) in which the matrix is partitioned as 16, 64, and 128. In adition we use two settings for PaToH: *default* and *quality*. The latter is PaToH optimized to obtain a better quality of the partitioning, but this can lead to a slower execution time.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

5.1. Enhanced versus other reordering algorithms: Results with SuperLU. In our first set of tests we use LU factorization with partial pivoting implemented in the SuperLU solver [15]. SuperLU uses threshold partial pivoting, with a preference for the diagonal element. We report results for SuperLU with a threshold of 1 that implements LU with partial pivoting in which at each step of factorization the element of maximum magnitude in the current column of L is used as pivot. We have also performed experiments with different values, as, for example, a threshold of 0.1, and a similar behavior of the algorithms has been observed.

To evaluate enhanced HUND, two preprocessing steps presented in section 3.4 are performed before the LU factorization with partial pivoting. First, the matrix is reordered using a HUND heuristic. Second, the matrix is reordered using the CCOLAMD algorithm, as presented in [11], and based on a postorder traversal of the row merge tree [22] of the diagonal blocks. After these two preprocessing steps, the LU factorization with partial pivoting of SuperLU is called. For the other reordering algorithms we do not use the second preprocessing step; that is, only the fill-reducing ordering is applied.

Table 5.2 displays for each reordering algorithm tested the fill-in; that is, the ratio of the nnzs of L and U to the nnzs of A. The cases represented in the table by "-" mean that SuperLU failed due to too much fill-in generated, and hence a memory

TABLE 5.2

Fill in the factors L and U, computed as $nnz(L+U-I)/nnz(A)$ obtained by different fill-reducing strategies with SuperLU using a threshold of value 1. The best result among COLAMD, AMD, METIS *on A^T+A,* METIS *on A^TA, and enhanced* HUND*-default is marked in boldface for each matrix.*

#	COLAMD	AMD	METIS (A^T+A)	METIS (A^TA)	Enhanced HUND-default			Enhanced HUND-quality		
					$k=16$	$k=64$	$k=128$	$k=16$	$k=64$	$k=128$
1	7.5	**5.7**	6.0	8.3	8.5	8.5	8.3	8.7	8.3	8.2
2	**13.6**	44.2	27.7	15.3	15.5	15.4	15.6	15.3	15.2	15.5
3	1.1	**1.0**	**1.0**	1.1	11.4	11.5	11.6	11.4	11.5	11.0
4	41.7	91.3	57.1	32.6	30.9	**30.2**	31.4	30.7	30.8	30.0
5	17.8	312.8	293.6	**16.8**	21.9	17.6	17.5	20.7	17.7	16.7
6	**4.0**	21.5	13.6	4.7	4.8	4.8	4.8	4.9	5.0	4.8
7	7.9	23.6	15.5	**7.0**	7.3	7.7	7.4	7.2	6.8	6.4
8	23.0	**7.9**	68.5	39.3	69.1	12.7	31.4	26.3	37.1	10.6
9	**3.5**	8.7	21.5	4.0	**3.5**	3.7	3.7	3.5	3.7	3.8
10	104.0	–	227.1	66.7	87.1	65.4	**64.8**	87.0	65.1	62.8
11	8.2	110.8	180.1	–	40.0	8.4	**5.2**	38.2	14.0	4.0
12	18.7	100.4	69.9	19.4	19.1	17.3	**17.0**	17.1	16.8	17.1
13	**4.8**	22.1	33.5	5.3	5.0	5.1	5.1	5.0	5.2	5.1
14	54.4	53.1	52.5	32.9	34.4	**32.2**	33.2	34.5	32.1	32.9
15	13.7	75.0	34.6	14.4	13.4	13.7	**12.8**	12.9	13.1	12.8
16	116.1	–	122.9	54.7	51.3	48.9	**47.3**	49.8	48.0	47.4
17	**25.0**	46.3	25.5	27.7	32.0	30.1	30.1	31.4	29.5	29.7
18	14.7	–	34.0	**13.0**	15.2	14.1	13.6	14.6	12.6	12.9
19	67.8	66.9	68.1	43.2	44.2	**42.5**	44.9	43.4	42.6	42.6
20	**5.6**	11.5	7.7	7.1	6.2	6.2	6.3	6.1	6.0	6.2
21	**4.6**	42.2	24.0	4.9	4.9	5.6	5.1	4.8	5.7	5.1
22	81.1	–	87.1	**56.6**	65.6	63.8	63.9	66.3	63.4	48.9
23	16.7	9.9	**8.9**	13.5	16.2	19.4	16.9	15.9	17.9	16.5
24	**13.3**	–	–	20.9	21.4	25.6	22.1	20.3	19.4	20.5
25	**4.7**	43.2	24.8	5.0	4.8	5.6	5.6	4.8	5.4	5.6
26	25.5	–	–	16.4	25.4	16.3	**16.1**	22.6	15.8	15.6
27	28.1	–	57.6	28.2	30.5	**27.0**	28.5	30.7	27.9	27.7

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

requirement that exceeded the limits of our computer. We focus on evaluating the performance of Hund based on PaToH with default parameter settings (Hund-default in the tables) and a varying number of partitions $k = 16, 64, 128$. We also present results obtained by Hund based on PaToH with quality parameter settings (Hund-quality in the tables). As we will see later, Hund-quality has a longer execution than Hund-default. Hence in practice it could be used when several matrices with the same nonzero structure need to be factored, and the reordering algorithm is called only once.

We observe that for 11 matrices in our test set, one variant of enhanced Hund-default induced the least fill-in compared to the other state-of-the-art reordering algorithms Colamd, Amd, and Metis. For 12 other matrices, each of Amd and Colamd produced the best results, while Metis produced the best result for 6 matrices.

For most of the matrices, enhanced Hund-default produces comparable results when the number of partitions increases from 16 to 128. As displayed in Table 5.2, the fill-in has a large value between 30 and 70 for the matrices mark3jac020, zhao2, sinc12, mark3jac140sc, sinc15, and sinc18 (numbers 4, 10, 14, 16, 19, and 22). However, for these matrices enhanced Hund produced the best, or close to the best, results. The other reordering strategies lead generally to a larger number of fill-in elements. Colamd leads to a fill-in factor between 42 and 116, and Metis (A^TA) leads to a fill-in factor between 32 and 66. For these cases enhanced Hund-default significantly outperforms Colamd. Note that matrices sinc12, sinc15, and sinc18 come from the same application domain (single-material crack problem) as well as matrices mark3jac020 and mark3jac140sc (economic model).

We consider more in detail the fill-in the factors L and U obtained by the two global strategies in our tests, enhanced Hund (with $kparts = 128$) and Metis (applied on the structure of A^TA, which was better than $A + A^T$). For most of the matrices in our test set enhanced Hund leads to comparable results to Metis, and sometimes it outperforms Metis. The best result is obtained for matrix shermanACb, for which enhanced Hund leads to almost four times less fill than Metis.

Figure 5.1 displays a performance profile of four orderings: enhanced Hund-default with $kparts = 64$, Colamd, Metis with A^TA, and Metis with $A + A^T$. In a performance profile, a method will have a point (x, y) in the graph if for y percent of the problems in the test set, the method leads to a fill-in that is within a factor of x (or better) of the fill-in found by the best method for those problems. We can notice that enhanced Hund provides a robust ordering, being either the best or within 10% of the best performance.

5.2. Enhanced Hund versus other reordering algorithms: Results with UMFPACK. UMFPACK is a right-looking multifrontal method which factorizes a sparse matrix using a sequence of frontal matrices, where each frontal matrix holds a contiguous set of pivot rows and columns. The ordering strategy in UMFPACK combines a fill-reducing symbolic preordering with adjustments made during numeric factorization.

As the first step of fill-reducing ordering, all pivots with zero Markowitz cost (referred to as *singletons*) are removed from the matrix. These correspond to the leading and trailing 1-by-1 blocks from a permutation to block triangular form. After singletons are removed, the remaining matrix is permuted to reduce fill-in in the LU factorization.

In UMFPACK's *unsymmetric strategy*, Hund (followed by CColamd on local blocks), Colamd, or Metis (applied to A^TA) is used to order the columns. In the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

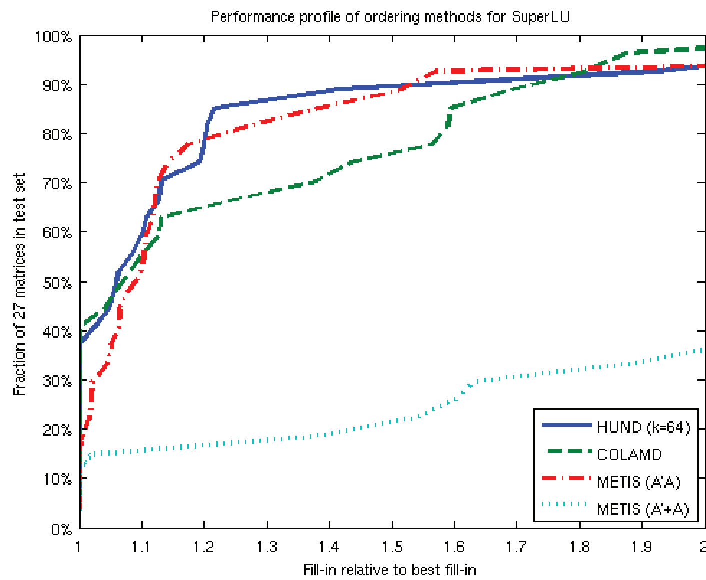

FIG. 5.1. *Performance profile of ordering methods with SuperLU. The performance is better closer to* 1.

symmetric strategy, AMD or METIS (applied to $A + A^T$) is used to order both the rows and columns, and a preference is given to the diagonal when selecting pivots. UMFPACK has an *automatic strategy*, which examines the matrix and tries to select the best strategy; we do not use this default method in the results in this paper. Note that we do not use MC64 to permute large entries on the diagonal of the matrix, since this step is not useful for UMFPACK's unsymmetric strategy.

The numerical factorization can revise the ordering computed by the symbolic preanalysis. Numerical threshold partial pivoting is used to select pivots within the pivot columns.

The size of each frontal matrix F is bounded by the frontal matrix that would arise in a sparse multifrontal QR factorization. Since this can be much larger than what is needed by an LU factorization, columns within each frontal matrix are reordered during numerical factorization to further reduce fill-in. This column reordering is only performed for the unsymmetric strategy; it is not performed by the symmetric strategy. Since UMFPACK is a right-looking method, it can consider the sparsity of a candidate pivot row when deciding whether or not to select it. This can be a key advantage over left-looking methods such as SuperLU [15]. Left-looking methods cannot consider the sparsity of candidate pivot rows, since the matrix to the right of the pivot column has not yet been updated when the pivot row is selected.

There are thus three primary differences between UMFPACK and SuperLU which affect the results presented in this paper:

1. UMFPACK removes singletons prior to factorization; SuperLU does not.
2. UMFPACK's unsymmetric strategy revises its column orderings within each frontal matrix to reduce fill-in; SuperLU does not revise the column orderings with each supercolumn.
3. UMFPACK can select a pivot row based on sparsity; SuperLU cannot.

Complete results are shown in Table 5.3. Figure 5.2 displays a performance profile of just four of the unsymmetric orderings (enhanced HUND-default with $kparts = 64$,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 5.3
Fill in the factors L and U, computed as $nnz(L+U-I)/nnz(A)$, using UMFPACK.

#	COLAMD	AMD	METIS (A^T+A)	METIS (A^TA)	Enhanced HUND $k=16$	$k=64$	$k=128$
1	**11.5**	40.5	23.8	14.5	14.1	14.1	14.1
2	14.5	134.4	125.0	**14.0**	14.9	14.5	14.6
3	1.0	1.0	1.0	1.0	1.0	**1.0**	1.0
4	**2.5**	4.0	11.7	2.8	2.7	2.9	2.9
5	6.9	**5.7**	6.0	7.7	7.3	7.4	7.3
6	32.9	45.0	36.9	29.3	26.1	**25.4**	25.8
7	**3.0**	13.0	9.5	3.8	3.8	3.7	3.6
8	**1.0**	2.0	2.1	**1.0**	**1.0**	**1.0**	**1.0**
9	78.7	47.2	**28.4**	55.7	57.5	53.0	52.0
10	**1.9**	2.3	2.8	2.3	1.9	2.0	2.0
11	4.4	**4.1**	4.9	4.1	4.3	4.2	4.3
12	8.8	**7.6**	9.5	10.7	8.9	8.8	8.9
13	**3.0**	4.7	13.0	3.6	3.4	3.5	3.5
14	38.3	36.5	35.4	**20.6**	32.7	21.1	21.4
15	79.6	63.9	42.0	44.7	43.5	41.0	**40.9**
16	10.2	-	37.8	11.6	10.1	**9.9**	10.2
17	22.0	**16.9**	17.5	24.9	25.7	26.3	26.2
18	49.9	-	46.5	29.9	42.5	**26.4**	40.2
19	6.8	14.4	11.6	8.5	**6.6**	7.4	8.4
20	**4.6**	9.8	6.1	5.6	5.1	5.4	5.1
21	3.8	26.8	20.9	4.2	**3.8**	4.0	4.2
22	60.7	-	-	**38.9**	-	-	-
23	**5.9**	-	14.4	8.3	8.0	7.9	8.3
24	3.8	28.4	-	4.2	**3.8**	4.0	4.1
25	14.3	9.4	**8.9**	11.6	13.6	13.1	13.6
26	20.2	56.1	-	**11.2**	12.6	12.0	11.7
27	23.7	-	25.8	25.6	24.2	23.7	**23.0**

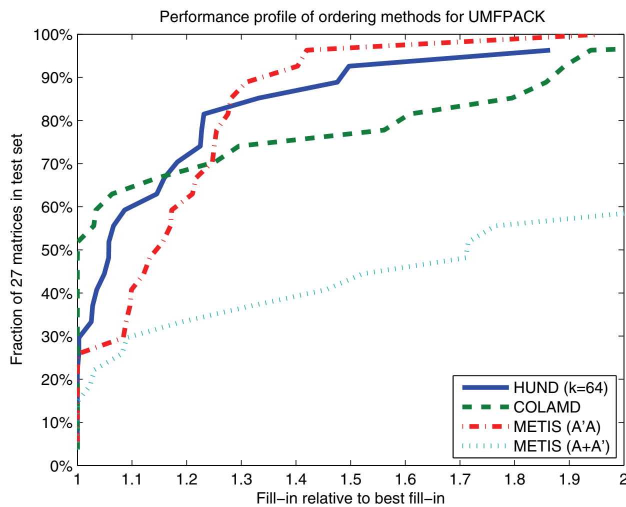

FIG. 5.2. *Performance profile of ordering methods with UMFPACK. The performance is better closer to 1.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Colamd, Metis with A^TA, and Metis with $A+A^T$). Overall, enhanced Hund provides a robust ordering with a performance profile superior to both Colamd and Metis. Notice that for about two-thirds of the matrices, the performance of enhanced Hund ($k = 64$) is within 10% of the best performance.

5.3. Quality of the partitioning. In this section we discuss the separators obtained during Hund's unsymmetric nested dissection. These separators are important because they tend to have an impact on the fill in the factors L and U as well as on the suitability of the reordered matrix for parallel execution. Fill in the factors can occur in the separators and in the diagonal blocks; hence the smaller the separator size, the fewer nonzeros should be obtained in the factors. In a parallel execution, the communication will incur during the factorization of the separators. Hence the communication will be reduced for separators of a smaller size.

We summarize here the more detailed results presented in the technical report [21] for Hund-default with $k = 128$. We discuss the size of the separators corresponding to the first three levels of unsymmetric nested dissection. The results for the other levels were in general less than 1%.

For most of the matrices, the separators obtained by Hund-default are of small size and contain less than 5% of the columns of the matrix (very often less than 1%).

However, for six matrices in our test set, the number of columns and the nnzs in the separators of the first and the second level are very large. For example, for matrix mult_dcop_03 (number 11), around 40% of the columns are in the first level separator, and an average of 11.5% of the columns are in the second level separator. Matrices shermanACb, sinc12, sinc15, and sinc18 (numbers $8, 14, 19, 22$) have more than 15% of the columns in the first level separator. As already previously observed and as reported in Table 5.2, for matrices in sinc family, enhanced Hund leads to a high amount of fill in the factors L and U. This observation shows that the size of the separator has an important impact on the quality of the reordering, that is, the nnzs in the factors L and U.

However, this is not always true. For example, the matrices zhao2 and mark3jac140sc (numbers 10, 16) have separators of small size. But the fill in the factors L and U is high: 64 for zhao2 and 47 for mark3jac140sc.

5.4. Runtime of enhanced Hund. Table 5.4 displays the execution time of each reordering algorithm tested. The execution times displayed for Metis include the time for forming $A^T + A$ and A^TA, respectively. The execution times displayed for enhanced Hund include the hypergraph partitioning with PaToH and the local ordering of diagonal blocks with CColamd. As it could be expected, the execution time of enhanced Hund increases with increasing number of partitions. Enhanced Hund based on PaToH with quality parameter settings (enhanced Hund-quality in the table) is usually between two and three times slower than enhanced Hund with default parameter settings (enhanced Hund-default in the table) for a same number of partitions. In this discussion, we focus on the results obtained by enhanced Hund default.

For the matrices in our test set, Colamd and Amd are faster than Metis and enhanced Hund. Comparing the global approaches' algorithms suitable for sparse LU factorization with partial pivoting of unsymmetric matrices, we can note that for smaller matrices, Metis on A^TA and enhanced Hund-default with 16 partitions have comparable execution times. However, for larger matrices Metis on A^TA becomes slower than enhanced Hund-default. In these cases, the computation of A^TA becomes an important part of the ordering time. This can be observed in our test set starting

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 5.4

Ordering time in seconds of COLAMD, AMD, METIS *(applied on $A^T + A$ and A^TA), and enhanced* HUND. *The best time is marked in boldface for each matrix. Enhanced* HUND *is tested with two different settings of PaToH: default and quality (enhanced* HUND *default and enhanced* HUND *quality). The number of partitions is varied as $k = 16, 64, 128$.*

#	COLAMD	AMD	METIS	METIS	Enhanced HUND default			Enhanced HUND quality		
			$(A^T + A)$	(A^TA)	$k = 16$	$k = 64$	$k = 128$	$k = 16$	$k = 64$	$k = 128$
1	0.02	**0.00**	0.03	0.05	0.12	0.29	0.42	0.43	1.04	1.33
2	**0.02**	0.02	0.05	0.12	0.16	0.29	0.38	0.70	1.25	1.49
3	**0.00**	0.02	0.15	0.17	0.08	0.12	0.16	0.23	0.45	0.62
4	0.07	**0.05**	0.17	0.28	0.30	0.53	0.65	1.17	1.91	2.23
5	**0.02**	0.03	0.22	0.30	0.28	0.61	0.87	1.19	2.53	3.46
6	0.05	**0.02**	0.15	0.22	0.24	0.35	0.36	1.05	1.54	1.46
7	0.02	**0.00**	0.03	0.17	0.20	0.25	0.29	0.82	1.08	1.12
8	0.42	**0.07**	0.25	8.42	0.36	0.51	0.50	5.35	5.78	5.62
9	0.08	**0.07**	0.42	0.60	0.41	0.72	0.86	1.40	2.79	3.19
10	**0.10**	–	0.57	0.88	0.83	1.37	1.72	4.06	6.02	7.76
11	0.12	**0.12**	0.32	–	1.25	1.39	1.49	14.27	13.84	13.21
12	0.55	**0.38**	0.75	1.40	1.13	1.35	1.56	3.45	4.55	5.26
13	0.20	**0.15**	1.00	1.67	0.63	1.27	1.57	2.14	4.70	6.28
14	**0.08**	0.17	0.47	0.75	0.52	0.56	0.63	1.73	1.96	1.83
15	**0.13**	0.15	0.72	1.08	0.84	1.08	1.26	2.43	3.78	4.49
16	**0.47**	–	1.47	2.55	1.74	2.47	2.87	6.83	10.12	11.55
17	0.73	**0.20**	0.65	1.80	1.49	2.25	2.39	4.21	6.72	8.46
18	**0.07**	–	0.20	0.63	0.73	1.02	1.20	2.32	3.40	4.31
19	**0.17**	0.33	0.83	1.52	0.90	0.99	1.06	2.12	3.19	2.76
20	1.02	**0.78**	1.98	4.53	2.45	3.19	3.26	5.53	8.11	9.22
21	0.37	**0.22**	1.73	2.80	1.39	2.36	2.85	4.13	7.61	9.96
22	**0.32**	–	1.32	3.12	1.49	1.72	1.82	5.97	4.06	4.53
23	0.68	0.22	1.67	3.23	1.88	3.08	3.95	5.86	11.79	15.63
24	**0.75**	0.78	–	15.07	2.86	3.90	4.29	7.13	11.66	14.58
25	0.75	**0.48**	3.75	6.18	2.46	3.84	5.11	6.08	11.39	16.95
26	**0.52**	–	2.37	17.02	5.92	6.91	7.75	12.99	16.25	17.58
27	2.67	–	**2.55**	8.55	6.78	8.16	8.58	15.06	20.61	21.91

with matrix SINC15. For example, for matrix AV41092, METIS on A^TA is almost 3 times slower than enhanced HUND-default with 16 partitions.

6. Conclusions and future work. We have presented a new ordering algorithm (HUND) for unsymmetric sparse matrix factorization, based on hypergraph partitioning and unsymmetric nested dissection. To enhance performance, we proposed a hybrid method that combines the nested dissection with local reordering. Our method allows partial pivoting without destroying sparsity. We have tested the method using SuperLU and UMFPACK, two well-known partial pivoting LU codes. Empirical experiments show that our method is highly competitive with existing ordering methods. In particular, it is robust in the sense that for about two-thirds of the matrices in our study it performs close to the best of all the other existing methods (within 10%). Thus, it is a good choice as an all-purpose ordering method.

The HUND method was designed for parallel computing, though we evaluated only it in serial here. The recursive top-down design allows coarse-grain parallelism, as opposed to local search methods like AMD and COLAMD. For symmetric systems, nested dissection ordering is considered superior for large systems, and it is reasonable to expect the same holds for unsymmetric systems. The most expensive part of enhanced HUND is hypergraph partitioning, which can be done efficiently in parallel using the Zoltan toolkit [16]. The matching for strong diagonal can also be performed

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

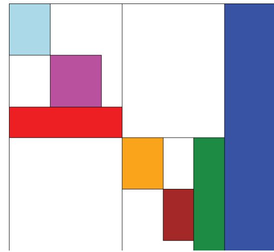

FIG. 6.1. *Example of variable direction (row or column) hybrid method. The top level separator is a column separator (rightmost block). The first (left) subproblem has a row separator, while the second (right) subproblem has a column separator.*

in parallel [36], though no parallel MC64 is yet available. Local reordering can be done locally in serial. Thus, our approach is well suited for fully parallel solvers that aim at being time and memory scalable [25, 35].

There are several directions for future work. HUND uses a column separator approach based on the column-net hypergraph model, where rows are vertices. The first direction of research is to use other options, as to use the row-net hypergraph model, where columns are vertices. The method will work as before, except now we find row separators instead of column separators. The row separator approach is advantageous when the row separator is smaller than the column separator. However, row permutations can now destroy the sparsity structure. This variation is thus not suitable for partial pivoting with row interchanges (though partial pivoting with column interchanges would be fine).

Since the best variation (row or column) depends on the matrix structure, an intriguing idea is to combine these two methods. The idea is to try both partitioning methods for every bisection and pick the best. This gives a recursive decomposition that uses a combination of row and column separators, and it is illustrated in Figure 6.1. This row-or-column hybrid method is also used in the Mondriaan method for sparse matrix partitioning [38]. Obtaining a strong diagonal is a bit more difficult with the hybrid method. As usual, we compute a matching in the bipartite graph, but it is not obvious how to apply this as a permutation. A pure row-or-column permutation of the entire matrix will ruin the sparsity structure. Instead, parts of the matrix should be permuted by columns and other parts by rows. We omit the details here.

A second direction is to study hybridization with other local (greedy) ordering methods, in particular, the recent unsymmetric methods by Amestoy, Li, and Ng [2] and Amestoy, Li, and Pralet [3].

A third direction of research is the usage of hypergraph partitioning in a parallel algorithm for performing the LU factorization of sparse matrices. It will be interesting to study the effect of the ordering on the performance of existing parallel solvers such as SuperLU_DIST [35] or MUMPS [4]. We note that MUMPS favors symmetric structures. Since HUND can destroy the symmetry for matrices that have a nearly symmetric structure, its usage might not be beneficial in the context of MUMPS. In addition, our goal is to design an algorithm that exploits the form of the matrix obtained after hypergraph partitioning and that uses tournament pivoting, a new stable pivoting strategy introduced in [23, 24] that reduces the amount of communication performed in parallel LU factorization.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] P. R. Amestoy, X. S. Li, and E. G. Ng, *Diagonal Markowitz scheme with local symmetrization*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 228–244.

[3] P. R. Amestoy, X. S. Li, and S. Pralet, *Unsymmetric ordering using a constrained Markowitz scheme*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 302–327.

[4] P. Amestoy, I. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 501–520. Special issue on Domain Decomposition and Parallel Computing.

[5] C. Aykanat, A. Pinar, and U. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput., 25 (2004), pp. 1860–1879.

[6] I. Brainman and S. Toledo, *Nested-dissection orderings for sparse LU with partial pivoting*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 998–1012.

[7] U. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Syst., 10 (1999), pp. 673–693.

[8] U. V. Çatalyürek and C. Aykanat, *PaToH: Partitioning tool for hypergraphs, User's Guide*, 1999.

[9] U. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, *Hypergraph-partitioning-based Fill-reducing Ordering*, Technical report OSU-BMI-TR-2009-n02, Ohio State University, 2009, submitted for publication.

[10] U. V. Çatalyürek, *Hypergraph Models for Sparse Matrix Partitioning and Reordering*, Ph.D. thesis, Bilkent University, Computer Engineering and Information Science, Ankara, Turkey, 1999.

[11] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, *Algorithm* 887: *CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, 35 (2008), pp. 1–14.

[12] T. A. Davis, J. R. Gilbert, S. Larimore, and E. Ng, *Algorithm* 836: *COLAMD, a column approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 377–380.

[13] T. A. Davis, *Algorithm* 832: *UMFPACK, an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Software, 30 (2004), pp. 196–199.

[14] T. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, to appear.

[15] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[16] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, *Parallel hypergraph partitioning for scientific computing*, in Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS'06), IEEE, 2006.

[17] I. S. Duff and J. Koster, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 973–996.

[18] I. S. Duff and J. A. Scott, *A parallel direct solver for large sparse highly unsymmetric linear systems*, ACM Trans. Math. Software, 30 (2004), pp. 95–117.

[19] I. S. Duff and J. A. Scott, *Stabilized bordered block diagonal forms for parallel sparse solvers*, Parallel Comput., 31 (2005), pp. 275–289.

[20] A. George, *Nested dissection of a regular finite-element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[21] L. Grigori, E. G. Boman, S. Donfack, and T. Davis, *Hypergraph-Based Unsymmetric Nested Dissection Ordering for Sparse LU Factorization*, Technical report 6520, INRIA, Paris, 2008.

[22] L. Grigori, M. Cosnard, and E. Ng, *On the row merge tree for sparse LU factorization with partial pivoting*, BIT, 47 (2007), pp. 45–76.

[23] L. Grigori, J. W. Demmel, and H. Xiang, *Communication avoiding Gaussian elimination*, in Proceedings of the ACM/IEEE SC08 Conference, 2008.

[24] L. Grigori, J. W. Demmel, and H. Xiang, *CALU: A Communication Optimal LU Factorization Algorithm*, Technical report UCB-EECS-2010-29, University of California, Berkeley, CA, 2010.

[25] L. Grigori, J. Demmel, and X. Li, *Parallel symbolic for sparse LU with static pivoting*, SIAM J. Sci. Comput., 29 (2007), pp. 1289–1314.

[26] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, in Proceedings of the Supercomputing '95, ACM, 1995.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[27] B. Hendrickson and E. Rothberg, *Improving the run time and quality of nested dissection ordering*, SIAM J. Sci. Comput., 20 (1998), pp. 468–489.
[28] HSL, *A Collection of Fortran Codes for Large Scale Scientific Computation*, http://www.cse.clrc.ac.uk/nag/hsl/.
[29] Y. Hu, K. Maguire, and R. Blake, *A multilevel unsymmetric matrix ordering algorithm for parallel process simulation*, Comput. Chem. Eng., 23 (2000), pp. 1631–1647.
[30] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, *Multilevel hypergraph partitioning: Applications in VLSI domain*, IEEE Trans. VLSI Syst., 7 (1999), pp. 69–79.
[31] G. Karypis and V. Kumar, Metis*: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices—verstion* 4.0, http://www-users.cs.umn.edu/karypis/metis.
[32] 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.
[33] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.
[34] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.
[35] X. S. Li and J. W. Demmel, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Math. Software, 29 (2003), pp. 110–140.
[36] F. Manne and R. Bisseling, *A parallel approximation algorithm for the weighted maximum matching problem*, in Proceedings of the Seventh International Conference on Parallel Processing and Applied Mathematics (PPAM 2007), 2007.
[37] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management, 3 (1957), pp. 255–269.
[38] B. Vastenhouw and R. H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, SIAM Rev., 47 (2005), pp. 67–95.
[39] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Algebr. Discrete Methods, 2 (1981), pp. 77–79.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.