

Frontal Solvers for Process Engineering: Local Row Ordering Strategies

Kyle V. Camarda
Department of Chemical Engineering
University of Illinois
600 S. Mathews Avenue
Urbana, IL 61801 USA

Mark A. Stadtherr¹
Department of Chemical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA

(revised, May 1997)

¹Author to whom all correspondence should be addressed

Abstract

The solution of chemical process simulation and optimization problems on today's high performance supercomputers requires algorithms that can take advantage of vector and parallel processing when solving the large, sparse matrices that arise. The frontal method can be highly efficient in this context due to its ability to make use of vectorizable dense matrix kernels on a relatively small frontal matrix in the innermost loop of the computation. However, the ordering of the rows in the coefficient matrix strongly affects size of the frontal matrix and thus the solution time. If a poor row ordering is used it may make the frontal method uncompetitive with other methods. We describe here a graph theoretical framework for identifying suitable row orderings that specifically addresses the issue of frontal matrix size. This leads to local, heuristic methods which aim to limit frontal matrix growth in the row and/or column dimensions. Results on a wide range of test problems indicate that improvements in frontal solver performance can often be obtained by the use of a restricted minimum column degree heuristic, which can be viewed as a variation of the minimum degree heuristic used in other contexts. Results also indicate that the natural unit-block structure of process simulation problems provides a quite reasonable ordering.

1 Introduction

The solution of realistic, industrial-scale simulation and optimization problems is computationally very intense, and may require the use of high performance computing technology to be done in a timely manner. For example, Zitney *et al.* (1995) described a dynamic simulation problem at Bayer AG requiring 18 hours of CPU time on a CRAY C90 supercomputer when solved with the standard implementation of SPEEDUP (Aspen Technology, Inc.). To better use this leading edge technology in process simulation and optimization requires the use of techniques that efficiently exploit vector and parallel processing. Since most current techniques were developed for use on conventional serial machines, it is often necessary to rethink problem solving strategies in order to take full advantage of supercomputing power. For example, by using a different linear equation solving algorithm and addressing other implementation issues, Zitney *et al.* (1995) reduced the time needed to solve the Bayer problem from 18 hours to 21 minutes.

In the Bayer problem, as in most other industrial-scale problems, the solution of large, sparse systems of linear equations is the single most computationally intensive step, requiring over 80% of the total simulation time in some cases. Thus, any reduction in the linear system solution time will result in a significant reduction in the total simulation time. The matrices that arise, however, do not have any of the desirable properties, such as numerical or structural symmetry, positive definiteness, and bandedness often associated with sparse matrices, and usually exploited in developing efficient parallel/vector algorithms. Recently, an implementation of the frontal method (Zitney, 1992; Zitney and Stadtherr, 1993; Zitney *et*

al., 1995), developed at the University of Illinois and later extended at Cray Research, Inc., has been described that is designed specifically for use in the context of process simulation. This solver (FAMP) has been incorporated in CRAY implementations of popular commercial codes, such as ASPEN PLUS, SPEEDUP (Aspen Technology, Inc.), and NOVA (Dynamic Optimization Technology Products, Inc.). A parallel version (PFAMP) has now also been developed (Mallya *et al.*, 1997) that better exploits multiprocessing.

FAMP is effective on parallel/vector machines since most of the computations involved can be performed using efficiently vectorized dense matrix kernels on relatively small frontal matrices. However, the performance of the frontal method is strongly dependent on the ordering of the rows in the matrix, since this ordering determines the size of the frontal matrix at each step of frontal elimination. For symmetric or banded matrices, ordering algorithms that attempt to minimize the matrix bandwidth or profile are effective in connection with the frontal method (e.g., Duff *et al.*, 1989), but for the highly asymmetric matrices that arise in process simulation, these techniques are inappropriate. Fortunately, the natural unit-block structure of process engineering problems can often provide a reasonable ordering, which has led to the many successful applications to date of the frontal method on such problems. On many problems, however, this natural ordering can clearly be improved. Thus, there is a need to consider reordering strategies that specifically address the needs of the frontal method in this context. In this paper we develop, in a graph theoretical framework, a general methodology for identifying suitable row orderings for frontal elimination applied to highly asymmetric matrices. Local heuristic techniques that aim to reduce frontal matrix

size are identified. Results on a wide variety of problems indicate that improvements in the performance of the frontal method can often be obtained through the use of such techniques.

2 Background

Consider the solution of a linear equation system $Ax = b$, where A is a large sparse $n \times n$ matrix and x and b are column vectors of length n . While iterative methods can be used to solve such systems, the reliability of such methods is questionable in the context of process simulation (Cofer and Stadtherr, 1996). Thus we concentrate here on direct methods. Generally such methods can be interpreted as an LU factorization scheme in which A is factored $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. Thus, $Ax = (LU)x = L(Ux) = b$, and the system can be solved by a simple forward substitution to solve $Ly = b$ for y , followed by a back substitution to find the solution vector x from $Ux = y$.

The frontal elimination scheme used here is an LU factorization technique that was originally developed to solve the banded matrices arising in finite element problems (Irons, 1970; Hood, 1976). The original motivation was, by limiting computational work to a relatively small *frontal matrix*, to be able to solve problems on machines with small core memories. Today it is widely used for finite element problems on parallel/vector supercomputers because, since the frontal matrix can be treated as dense, most of the computations involved can be performed by using very efficient vectorized dense matrix kernels.

Process simulation matrices are not banded, but due to the unit-stream structure of

the problem, they may be nearly block-banded. Recycle and feedforward streams cause off-band blocks to occur, as can design specifications. Thus, Stadtherr and Vegeais (1985) extended the idea of frontal elimination to the solution of process simulation problems on supercomputers, and later (Vegeais and Stadtherr, 1990) demonstrated its potential. As noted above, an implementation of the frontal method developed specifically for use in the process simulation context has been described by Zitney (1992), Zitney and Stadtherr (1993), and Zitney *et al.* (1995), and is now incorporated in supercomputer versions of popular process simulation and optimization codes.

The frontal elimination scheme can be outlined briefly as follows:

1. Assemble a row into the frontal matrix.
2. Determine if any columns are fully summed in the frontal matrix. A column is fully summed if it has all of its nonzero elements in the frontal matrix.
3. If there are fully summed columns, then perform partial pivoting in those columns, eliminating the pivot rows and columns and doing an outer-product update on the remaining part of the frontal matrix.

This procedure begins with the assembly of row 1 into the initially empty frontal matrix, and proceeds sequentially row by row until all are eliminated, thus completing the LU factorization. To be more precise, it is the LU factors of the permuted matrix PAQ that have been found, where P is a row permutation matrix determined by the partial pivoting, and Q is a column permutation matrix determined by the order in which the columns become

fully summed. Thus the solution to $Ax = b$ is found as the solution to the equivalent system $PAQQ^T x = LUQ^T x = Pb$, which is solved by forward substitution to solve $Ly = Pb$ for y , back substitution to solve $Uw = y$ for w , and finally the permutation $x = Qw$. To simplify notation, the permutation matrices will henceforth not be shown explicitly.

To see this in mathematical terms, consider the submatrix $A^{(k)}$ remaining to be factored after the $(k - 1)$ -th pivot:

$$A^{(k)} = \left[\begin{array}{cc|c} F^{(k)} & 0 & \\ \hline A_{ps}^{(k)} & A_{ns}^{(k)} & \end{array} \right]. \quad (1)$$

Here $F^{(k)}$ is the frontal matrix. The subscript ps in $A_{ps}^{(k)}$ indicates that it contains columns that are partially summed (some but not all nonzeros in the frontal matrix) and the subscript ns in $A_{ns}^{(k)}$ indicates that it contains columns that are not summed (no nonzeros in the frontal matrix). If a stage in the elimination process has been reached at which all remaining columns have nonzeros in the frontal matrix, then $A_{ns}^{(k)}$ and the corresponding zero submatrix will not appear in Eq. (1). Assembly of rows into the frontal matrix then proceeds until $g_k \geq 1$ columns become fully summed:

$$A^{(k)} = \left[\begin{array}{cc|c} \bar{F}_{11}^{(k)} & \bar{F}_{12}^{(k)} & 0 \\ \bar{F}_{21}^{(k)} & \bar{F}_{22}^{(k)} & 0 \\ \hline 0 & \bar{A}_{ps}^{(k)} & \bar{A}_{ns}^{(k)} \end{array} \right]. \quad (2)$$

$\bar{F}^{(k)}$ is now the frontal matrix and its submatrices $\bar{F}_{11}^{(k)}$ and $\bar{F}_{21}^{(k)}$ comprise the columns that

have become fully summed, which are now eliminated using rows chosen during partial pivoting and which are shown as belonging to $\bar{F}_{11}^{(k)}$ here. This amounts to the factorization $\bar{F}_{11}^{(k)} = L_{11}^{(k)}U_{11}^{(k)}$ of the order- g_k block $\bar{F}_{11}^{(k)}$, resulting in:

$$A^{(k)} = \left[\begin{array}{cc|c} L_{11}^{(k)}U_{11}^{(k)} & U_{12}^{(k)} & 0 \\ L_{21}^{(k)} & F^{(k+g_k)} & 0 \\ \hline 0 & A_{ps}^{(k+g_k)} & A_{ns}^{(k+g_k)} \end{array} \right] \quad (3)$$

where the new frontal matrix $F^{(k+g_k)}$ is the Schur complement $F^{(k+g_k)} = \bar{F}_{22}^{(k)} - L_{21}^{(k)}U_{12}^{(k)}$, which is computed using an efficient full-matrix outer-product update kernel, $A_{ps}^{(k+g_k)} = \bar{A}_{ps}^{(k)}$ and $A_{ns}^{(k+g_k)} = \bar{A}_{ns}^{(k)}$. Note that operations are done within the frontal matrix only. At this point $L_{11}^{(k)}$ and $L_{21}^{(k)}$ contain columns k through $k + g_k - 1$ of L and $U_{11}^{(k)}$ and $U_{12}^{(k)}$ contain rows k through $k + g_k - 1$ of U . The computed columns of L and rows of U are saved and the procedure continues with the assembly of the next row into the new frontal matrix $F^{(k+g_k)}$.

Note that the order in which the variables are eliminated, and thus the column ordering in the L and U factors, is determined by the order in which the columns become fully summed, which in turn is determined by the row ordering. For example, for the matrix in Figure 1(a), column 4 is the first to become fully summed as rows are assembled into the frontal matrix and is thus the first to be eliminated and the first column in the L and U factors. Columns 2 and 5 would follow next, and then columns 1, 3 and 6. For a different row order, say one in which rows 4 and 5 are switched, the column elimination order will be different, with in this case columns 2 and 5 now coming first, followed by column 4 and then columns 1, 3 and

6. Thus, the row ordering determines the ultimate column ordering and the two cannot be stipulated independently.

Note also the effect of the row ordering on the frontal matrix size, and thus on the number of operations needed to perform the outer-product update. Again referring to Figure 1(a), the first frontal matrix to be operated on occurs after column 4 becomes fully summed and thus will be 4×6 . On the other hand, if say rows 2 and 5 were ordered first, then column 2 would become fully summed after just these two rows entered the frontal matrix, and so the first frontal matrix to be operated on would be 2×3 .

For banded matrices, the frontal matrix will be relatively small and dense, with a maximum size of $b(2b - 1)$, where b is the semibandwidth of the matrix. However, for process simulation matrices, the frontal matrix may become relatively large and sparse. In this case, wasted operations are performed on zeros since dense matrix operations are used in the outer-product update, and thus a trade-off occurs between increased operation count and increased operation rate due to vectorization. Thus arises the need for a good row ordering to keep the size of the frontal matrix small. The natural unit-block structure of process simulation matrices can provide a reasonable ordering for this purpose. This structure will occur provided that the application code generating the matrix groups together the equations from each unit model, and that some effort is made to number adjacent units consecutively. Nevertheless, this natural ordering can clearly be improved in many cases. We consider here the extent to which it can be improved through the use of some simple heuristic techniques.

3 Reordering Methods

Due to its origin and continued wide use as a band or profile solver for finite element problems, most reordering methods that have been applied in connection with the frontal method are bandwidth or profile reduction techniques (e.g., Cuthill and McKee, 1969; Gibbs *et al.*, 1976; Everstine, 1979; Sloan, 1986; Duff *et al.*, 1989). Such techniques are designed for structurally symmetric matrices; however, if the structure of A is only slightly asymmetric, useful results may be obtained by applying the reordering technique to the structure of $A + A^T$. For highly asymmetric matrices, such as occur in process simulation, using the structure of $A + A^T$ cannot be expected to yield good results, as the number of additional nonzeros in $A + A^T$, indicating dependencies that are in fact nonexistent in the actual problem, will be very large, nearly as large as the number of nonzeros indicating actual dependencies in many cases.

For highly asymmetric problems, one approach that has been used (Zitney and Stadtherr, 1993) is to use techniques that aim to produce a nearly triangular form, with relatively few nonzero columns (spikes) protruding from the triangular part. Such techniques include the well-known P^4 algorithm (Hellerman and Rarick, 1972) and the techniques of Stadtherr and Wood (1984). These methods are designed to produce a nearly lower triangular form, while for the frontal method a nearly upper triangular form is desired. This can be obtained by simply reversing the row order (the column order is immaterial since it is determined from the row order, as discussed above). In using an ordering such as reverse- P^4 (r- P^4) only the spike columns will remain in the frontal matrix for more than one elimination step, and the

closer the nonzeros in the spike columns are to the diagonal the shorter their stay in the frontal matrix will be. In our experience, which is reflected in the results below, this sort of approach can produce a very good row ordering for the frontal method in some cases, but very a bad ordering in other cases. This is not surprising inasmuch as these reordering techniques were not developed with the frontal method in mind, but for fill reduction, and thus do not directly address the issue of frontal matrix size.

We now proceed to develop a graph theoretical framework that will allow us to directly address the issue of frontal matrix size in obtaining a row ordering. A bipartite graph representation of the matrix is used since this form allows an unsymmetric matrix without a full transversal to be represented. The bipartite graph $\mathcal{G} = (\mathcal{R}, \mathcal{C}, \mathcal{E})$ consists of a row vertex set \mathcal{R} , a column vertex set \mathcal{C} , and an edge set $\mathcal{E} = \{(r_i, c_j) \mid r_i \in \mathcal{R}, c_j \in \mathcal{C}, a_{ij} \neq 0\}$ corresponding to the non-zero elements in the matrix. A *net* is defined as a column vertex and its adjacent row vertices and the set of all nets is \mathcal{N} . The net $n_c \in \mathcal{N}$ corresponding to column $c \in \mathcal{C}$ is then $n_c = \text{Net}(c) = \{c \cup \text{Adj}(c)\}$, where $\text{Adj}(c) = \{r \in \mathcal{R} \mid (r, c) \in \mathcal{E}\}$. Thus, for example in the graph of Figure 1(b), corresponding to the matrix of Figure 1(a), $n_1 = \text{Net}(c_1) = \{c_1, r_1, r_6\}$, $n_2 = \text{Net}(c_2) = \{c_2, r_2, r_5\}$, etc. In these terms, one can construct a (symmetric) net-column occurrence matrix and a corresponding net-column graph $\mathcal{G}_{\mathcal{N}} = (\mathcal{N}, \mathcal{E}_{\mathcal{N}})$ in which edges and corresponding occurrence matrix entries represent the columns adjacent to each net in the row-column graph. That is, $\mathcal{E}_{\mathcal{N}} = \{(n_i, n_j), i \neq j \mid n_i \in \mathcal{N}, n_j \in \mathcal{N}, c_i \in \text{Adj}(n_j)\}$, where $\text{Adj}(n_j) = \{c \in \mathcal{C} \mid (r, c) \in \mathcal{E} \text{ for any } r \in n_j\}$. This is shown in Figure 2, which corresponds to the matrix and bipartite graph in Figure 1. Here, for

instance, since columns c_3 , c_4 , c_5 and c_6 are adjacent to net n_1 , there are edges $(3,1)$, $(4,1)$, $(5,1)$ and $(6,1)$ in $\mathcal{G}_{\mathcal{N}}$ and corresponding edges in the net-column occurrence matrix.

As discussed above, by specifying an initial row assembly ordering, one indirectly fixes the column elimination order. Conversely, if one specifies an initial column elimination order, that can be viewed as indirectly fixing the row assembly ordering, at least if one stipulates that only those additional rows necessary for the elimination of the next column in the column elimination order are in fact assembled into the frontal matrix. Since these are the rows belonging to the net associated with that column, we can thus view the row ordering problem indirectly as a net ordering problem. Thus, for instance in Figure 1(a), if column c_1 is eliminated first, rows r_1 and r_6 are the only rows that need be in the frontal matrix and so those rows (members of net n_1) are ordered first. Similarly, if c_2 is eliminated first then rows r_2 and r_5 (members of net n_2) would be ordered first. To then choose the net order, we consider the size of the frontal matrix needed to eliminate the corresponding column. If net n_1 is ordered first, then c_1 will be eliminated first and the first frontal matrix will be 2×5 (rows r_1 and r_6 ; columns c_1 , c_3 , c_4 , c_5 and c_6), but if net n_2 is ordered first, then c_2 is eliminated first and the first frontal matrix will be 2×3 (rows r_2 and r_5 ; columns c_2 , c_3 and c_5). Thus, at least in a local sense, ordering n_2 first is preferable. Note that for a chosen initial net n_j the rows in the frontal matrix constitute the set $\mathcal{R}_{F,j} = \{r \in \mathcal{R} \mid r \in n_j\}$ and the columns in the frontal matrix constitute the set $\mathcal{C}_{F,j} = \{c \in \mathcal{C} \mid c \in c_j \cup \text{Adj}(n_j)\}$. So, in this scheme, for an initial net n_j , the row dimension of the initial frontal matrix corresponds to $d_{\mathcal{G}}(c_j)$, the degree (number of adjacent vertices) of c_j in \mathcal{G} (i.e., the number of nonzeros

in column c_j), and the column dimension corresponds to one plus $d_{\mathcal{G}_N}(n_j)$, the degree of n_j in \mathcal{G}_N ($d_{\mathcal{G}_N}(n_j)$ corresponds to the number of entries in row n_j of the net-column occurrence matrix). After the first elimination is performed, the first net can be removed from \mathcal{G} and \mathcal{G}_N and the column and net degrees updated, after which $d_{\mathcal{G}}(c_j)$ will represent the increase in the row dimension and $d_{\mathcal{G}_N}(n_j)$ the *potential* increase in the column dimension (since some columns in $\mathcal{C}_{F,k}$ may already be in the frontal matrix and this is not reflected in the net degree) as each subsequent net n_k is ordered and corresponding column eliminated. By using information about the column and net degree to choose the net ordering, and thus the order in which the variables are eliminated, it is hoped that the size of the frontal matrix can be kept small throughout the elimination process.

3.1 RMCD ordering

To implement a net ordering scheme within this framework, an appropriate merit function based on column and net degrees must be chosen. The simplest choice would be to just use the column degree; thus the next net n_j in the ordering would be chosen so that $d_{\mathcal{G}}(c_j) = \min_k d_{\mathcal{G}}(c_k)$. This would locally minimize the growth in the row dimension of the frontal matrix. Thus arises the ubiquitous *minimum degree* ordering (e.g., Tinney and Walker, 1967; Rose, 1973; Duff and Reid, 1974; George and Liu, 1989; Davis *et al.*, 1996), which is widely used as a fill reducing ordering, especially for symmetric problems. This does not address at all growth in the column dimension of the frontal matrix. One simple way of doing this, to at least a small extent, within the context of minimum column degree is to

restrict the choice of the next net n_j to those whose column vertices are already in the frontal matrix (i.e., any column vertex whose degree has been updated). In this way there must be at least some overlap between the columns already in the frontal matrix and the set of columns $\mathcal{C}_{F,j}$ brought into the frontal matrix by net n_j . We refer to this as the restricted minimum column degree ordering (RMCD).

For example, consider the matrix in Figure 1(a). Here the initial column degrees are: $d_{\mathcal{G}}(c_1) = 2$, $d_{\mathcal{G}}(c_2) = 2$, $d_{\mathcal{G}}(c_3) = 4$, $d_{\mathcal{G}}(c_4) = 3$, $d_{\mathcal{G}}(c_5) = 3$ and $d_{\mathcal{G}}(c_6) = 3$. The first net chosen is n_1 since $d_{\mathcal{G}}(c_1) = \min_k d_{\mathcal{G}}(c_k) = 2$ (tiebreakers are discussed below; here we choose the net with lowest index). Thus we order rows $\mathcal{R}_{F,1} = \{r \in \mathcal{R} \mid r \in n_1\} = \{r_1, r_6\}$ first, and these rows together with columns $\mathcal{C}_{F,1} = \{c \in \mathcal{C} \mid c \in c_1 \cup \text{Adj}(n_1)\} = \{c_1, c_3, c_4, c_5, c_6\}$ form the first frontal matrix, which is 2×5 . The net and rows ordered are now removed from $\mathcal{G}_{\mathcal{N}}$ and \mathcal{G} , respectively, and the column degrees in \mathcal{G} updated, resulting in: $d_{\mathcal{G}}(c_2) = 2$, $d_{\mathcal{G}}(c_3) = 3$, $d_{\mathcal{G}}(c_4) = 2$, $d_{\mathcal{G}}(c_5) = 2$ and $d_{\mathcal{G}}(c_6) = 2$. The next net chosen is n_4 since c_4 is the first updated column with minimum degree (n_2 is not chosen since $d_{\mathcal{G}}(c_2)$ has not yet been updated). Thus, rows $\mathcal{R}_{F,4} = \{r \in \mathcal{R} \mid r \in n_4\} = \{r_3, r_4\}$ are ordered next. Potential new columns in the frontal matrix are $\mathcal{C}_{F,4} = \{c \in \mathcal{C} \mid c \in c_4 \cup \text{Adj}(n_4)\} = \{c_3, c_4, c_5, c_6\}$. However, all of these are already in the frontal matrix so there is no growth in its column dimension, resulting in a frontal matrix that is 3×5 . This procedure continues until all rows are ordered, resulting in the reordered matrix shown in Figure 3.

In implementing this, the matrix is first forward and backward triangularized (e.g., Stadtherr and Wood, 1984) as much as possible by removing and ordering any 1×1 ir-

reducible blocks that can be placed at the beginning or end of the new ordering. These rows do not need to be processed in the frontal matrix and thus need not be considered in applying RMCD or the other orderings discussed below. For the implementations described here, a full partitioning into block upper triangular form is not attempted.

3.2 MNA ordering

The RMCD ordering does not directly address the growth of the frontal matrix in the column dimension. To do this the net degree $d_{\mathcal{G}_N}(n_j)$ should be also included in the merit function since $d_{\mathcal{G}_N}(n_j)+1$ reflects the potential growth of the column dimension of the frontal matrix. One simple merit function that could be used is the product $\alpha(n_j) = d_{\mathcal{G}}(c_j)[d_{\mathcal{G}_N}(n_j)+1]$, which we refer to as the net “area”. The *minimum net area* (MNA) ordering chooses the next net n_j so that $\alpha(n_j) = \min_k \alpha(n_k)$. As a tiebreaker, the net whose degree was last updated is chosen, for reasons discussed above.

For example, consider again the matrix in Figure 1(a). The initial column degrees are as given above and the initial net degrees are: $d_{\mathcal{G}_N}(n_1) = 4$, $d_{\mathcal{G}_N}(n_2) = 2$, $d_{\mathcal{G}_N}(n_3) = 5$, $d_{\mathcal{G}_N}(n_4) = 4$, $d_{\mathcal{G}_N}(n_5) = 4$ and $d_{\mathcal{G}_N}(n_6) = 3$. Thus the initial net areas are: $\alpha(n_1) = (2)(5) = 10$, $\alpha(n_2) = (2)(3) = 6$, $\alpha(n_3) = (4)(6) = 24$, $\alpha(n_4) = (3)(5) = 15$, $\alpha(n_5) = (3)(5) = 15$ and $\alpha(n_6) = (3)(4) = 12$. The net with the minimum area is net n_2 so it is chosen first. Thus we order rows $\mathcal{R}_{F,2} = \{r \in \mathcal{R} \mid r \in n_2\} = \{r_2, r_5\}$ first, and these rows together with columns $\mathcal{C}_{F,2} = \{c \in \mathcal{C} \mid c \in c_2 \cup \text{Adj}(n_2)\} = \{c_2, c_3, c_5\}$ form the first frontal matrix, which is 2×3 . The net and rows ordered are now removed from \mathcal{G}_N and \mathcal{G} , respectively,

and the column and net degrees updated, resulting in: $d_{\mathcal{G}}(c_1) = 2$, $d_{\mathcal{G}}(c_3) = 2$, $d_{\mathcal{G}}(c_4) = 3$, $d_{\mathcal{G}}(c_5) = 1$ and $d_{\mathcal{G}}(c_6) = 3$ for the column degrees, and $d_{\mathcal{G}_N}(n_1) = 4$, $d_{\mathcal{G}_N}(n_3) = 4$, $d_{\mathcal{G}_N}(n_4) = 4$, $d_{\mathcal{G}_N}(n_5) = 3$ and $d_{\mathcal{G}_N}(n_6) = 3$ for the net degrees. So the new net areas are $\alpha(n_1) = (2)(5) = 10$, $\alpha(n_3) = (2)(5) = 10$, $\alpha(n_4) = (3)(5) = 15$, $\alpha(n_5) = (1)(4) = 4$ and $\alpha(n_6) = (3)(4) = 12$. Net n_5 has the minimum net area so it is chosen next. Thus, rows $\mathcal{R}_{F,5} = \{r \in \mathcal{R} \mid r \in n_5\} = \{r_1\}$ are ordered next. Potential new columns in the frontal matrix are $\mathcal{C}_{F,5} = \{c \in \mathcal{C} \mid c \in c_5 \cup \text{Adj}(n_5)\} = \{c_1, c_3, c_4, c_5\}$, of which only c_1 and c_4 are actually new. The frontal matrix size is now 2×4 . This procedure continues until all rows are ordered, resulting in the reordered matrix shown in Figure 4. As in the case of RMCD, backward and forward triangularization is done on the matrix before application of the MNA ordering.

4 Results and Discussion

A large number of test problems were used in this study to test the usefulness of different reordering strategies. Table 1 gives information about each test matrix, including the order, the number of nonzeros, the symmetry ratio, and the matrix source. The symmetry ratio s is the number of matched off-diagonal nonzeros (i.e., nonzeros $a_{ij}, i \neq j$, for which a_{ji} is also nonzero) divided by the total number of off-diagonal non-zeroes. Thus $s = 1$ indicates a completely symmetric pattern, while $s = 0$ shows complete asymmetry. Except for the last six problems listed, all the matrices arise from applications in process simulation, using SEQUEL (Zitney and Stadtherr, 1988), SPEEDUP (Aspen Technology, Inc.) and ASPEN

PLUS (Aspen Technology, Inc.), as well as three distillation problems (*west**) from the Harwell/Boeing collection. The last six matrices are also from the Harwell/Boeing collection, and represent a variety of applications outside of process simulation.

Each of the test matrices was reordered using r-P⁴, RMCD, and MNA. The reordered matrices were then solved using the frontal solver FAMP on one processor of a CRAY C90. Table 1 shows the time in seconds required to solve each linear system when the original, natural row order is used. Then for each reordering, the ratio ($\times 100$) of the time required to solve the reordered problem to the time required with the natural ordering is given. Thus, for instance, the figure of 72.1% in the RMCD column for problem *ngc* indicates a solution time of $(0.721)(0.0540) = 0.0389$ s when the RMCD ordering was used. The best performance on each problem is highlighted by putting it in bold face. A summary of the results is given schematically in Figure 5. This indicates, for each ordering, the average normalized solution time, which is computed as follows. For each problem and method the solution time is normalized by dividing by the solution time of the best method on that problem. This is then averaged over all problems to determine the average normalized solution time for each method. Thus, a figure of one for a method would indicate that the method was the best method on all problems. The last six problems in Table 1 (i.e., those that are not process simulation problems) were not included in the average.

Clearly the most attractive ordering overall was that provided by RMCD. On a large majority of the problems, it provided a significant, though usually not dramatic, improvement in solution time, and on those problems where it did not provide any improvement,

it did not result in a substantial increase in solution time. The r-P⁴ ordering performed very well in many cases and very poorly in others, reflecting our previous experience with this technique in this context. The performance of MNA was disappointing; it provided competitive reorderings in many cases but performed poorly in many cases as well. The difficulty appears to be that the net degree $d_{\mathcal{G}_N}(n_j)$ often provides a poor measure of the *actual* growth in the column dimension of the frontal matrix caused by selecting net n_j , because of significant overlap between the columns of $\mathcal{C}_{F,j}$ and those columns already in the frontal matrix. An important observation is that, as suggested above, for process simulation problems the natural ordering arising from the unit-block structure of the problem is in fact quite a good ordering.

4.1 Tiebreakers

In implementing the RMCD ordering there is frequently a need to break ties between two or more nets whose column vertices all have the same degree in \mathcal{G} . In the version of RMCD discussed above this was done by simply choosing the net with the lowest index. We have investigated various tiebreaking schemes with the usual result being that these variants have little effect on the overall effectiveness of the reordering methods. In some cases, however, it can make a substantial and unpredictable difference. For example, in the version RMCD1, ties are broken by taking the net whose column degree was most recently updated. In the large majority of the problems there is little difference in the performance of RMCD1 and RMCD. However, when applied to the *lms3937* problem, the solution time

using the RMCD1 ordering was 2.95 s, compared to 0.23 s when the RMCD ordering was used. In other cases, the situation is reversed and the RMCD1 ordering performs significantly better than RMCD. Similar results, namely little effect on most problems, but significant and unpredictable effects on others, were also observed when other tiebreakers were used, including the use of net degree, and the use of a one-step look ahead. This situation suggests that putting significant effort into tiebreaking strategies is not particularly useful. This is not surprising inasmuch as the reordering heuristics used here are strictly *local* and do not directly seek the best *global* ordering, but try to build a good ordering incrementally through a series of local decisions. A locally good decision made early in the reordering process may ultimately force a choice among poor alternatives later, thus leading to a poor global ordering.

4.2 Reordering Time

It is easy to spend a substantial amount of computation time in addressing the reordering problem described here, since globally it is a combinatorial optimization problem. Even applying simple heuristics such as RMCD is not inexpensive. The computationally intensive part of the RMCD strategy is the updating of the column degrees, just as in the minimum degree algorithms used in the context of fill reduction. In the fill reduction context, there has been considerable effort (e.g., Duff and Reid, 1983; Eisenstat *et al.*, 1981; George and Liu, 1980a,b; Liu, 1985) spent on reducing the work required to keep track of the degrees, and recently work (e.g., Gilbert *et al.*, 1992; Davis and Duff, 1997; Davis *et al.*, 1996) has

concentrated on using approximations of (generally upper bounds on) the degrees, in order to further reduce computational requirements. Our implementations do not employ these useful approximations and should not be considered as the best that could be achieved with respect to reordering time.

The reordering time needs to be assessed relative to the improvement in solution time it provides. Thus, we consider here the ratio of reordering time to improvement in solution time. Frequently in process simulation, as discussed further below, matrices with the same structure must be solved repeatedly. Thus, generally reordering time can be amortized over a number of solves (refactorizations). From this standpoint the ratio of reordering time to improvement in solution time can be interpreted as the number of solves over which the reordering must be amortized in order to break even. Thus we refer to this ratio as the breakeven number N_{BE} ; if the number of solves needed is more than N_{BE} , then the reordering has been worthwhile in terms of total computing time; if less than N_{BE} , then computing time could have been saved by not doing a reordering and using the natural order. For RMCD, on those cases in which a significant savings (say 25%) in solution time was obtained, the N_{BE} values are typically around two. For example, $N_{BE} = 2.3$ for problem *ngc* and 2.1 for *lhr*. As savings in solution time becomes less significant, the value of N_{BE} quickly increases, for example to 23.2 for *mult2* and 32.3 for *mpex2*, until finally when there is no improvement in solution time, this measure becomes meaningless.

5 Concluding Remarks

The results presented above demonstrate that the simple RMCD heuristic for row ordering can often provide a significant reduction in the time needed to solve process simulation matrices using the frontal method, and can do so relatively inexpensively. The cost of a good ordering must be weighed against the number of times the given simulation or optimization problem is going to be solved. Typically, if the effort is made to develop a large scale simulation or optimization model, then it is likely to be used a very large number of times, especially if it is used in an on-line, operations environment, as is today becoming increasingly common. Even if the need for matrix refactorizations can be made relatively infrequent, as in some dynamic simulations, the number of refactorizations done over the lifetime of a process model will still be very large. Thus, especially for on-line applications, the investment made to find a good ordering might have substantial long term paybacks, which might in fact justify the use of more expensive reorderings that address the issue from a global, rather than local, standpoint.

Acknowledgments – This work has been supported by the National Science Foundation under Grants DMI-9322682 and DMI-9696110. We also acknowledge the support of the National Center for Supercomputing Applications at the University of Illinois, Cray Research, Inc. and Aspen Technology, Inc.

References

- Cofer, H. N. and M. A. Stadtherr, Reliability of iterative linear solvers in chemical process simulation. *Comput. Chem. Engng*, **20**, 1123–1132 (1996).
- Cuthill, E. and J. McKee, Reducing the bandwidth of sparse symmetric matrices. In *Proc. ACM National Conference*, pages 157–172. Association for Computing Machinery, New York (1969).
- Davis, T. A., P. Amestoy, and I. S. Duff, An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, **17**, 886–905 (1996).
- Davis, T. A. and I. S. Duff, An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Appl.*, **18**, 140–158 (1997).
- Duff, I. S. and J. K. Reid, A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination. *Int. Inst. Math. Applic.*, **14**, 281–291 (1974).
- Duff, I. S. and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw.*, **9**, 302–325 (1983).
- Duff, I. S., J. K. Reid, and J. A. Scott, The use of profile reduction algorithms with a frontal code. *Int. J. Num. Meth. Eng.*, **28**, 2555–2568 (1989).
- Eisenstat, S. C., M. H. Schultz, and A. H. Sherman, Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM J. Sci. Stat. Comput.*, **2**, 225–237 (1981).

- Everstine, G. C., A comparison of three resquencing algorithms for the reduction of matrix profile and wavefront. *Int. J. Num. Meth. Eng.*, **14**, 837–853 (1979).
- George, A. and J. W. H. Liu, A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Softw.*, **6**, 337–358 (1980a).
- George, A. and J. W. H. Liu, A minimal storage implementation of the minimum degree algorithm. *SIAM J. Numer. Anal.*, **17**, 282–299 (1980b).
- George, A. and J. W. H. Liu, The evolution of the minimum degree ordering algorithm. *SIAM Review*, **31**, 1–19 (1989).
- Gibbs, N. E., W. G. Poole, and P. K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.*, **13**, 236–250 (1976).
- Gilbert, J. R., C. Moler, and R. Schreiber, Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, **13**, 333–356 (1992).
- Hellerman, E. and D. Rarick, The partitioned preassigned pivot procedure (P^4). In Rose, D. J. and R. A. Willoughby, editors, *Sparse Matrices and Their Applications*. Plenum Press, New York (1972).
- Hood, P., Frontal solution program for unsymmetric matrices. *Int. J. Numer. Meth. Engng*, **10**, 379 (1976).
- Irons, B. M., A frontal solution program for finite element analysis. *Int. J. Numer. Meth. Engng*, **2**, 5 (1970).

- Liu, J. W. H., Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Softw.*, **11**, 141–153 (1985).
- Mallya, J. U., S. E. Zitney, S. Choudhary, and M. A. Stadtherr, A parallel frontal solver for large scale process simulation and optimization. *AIChE J.*, **43**, 1032–1040 (1997).
- Rose, D. J., A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Read, R. C., editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York (1973).
- Sloan, S. W., An algorithm for profile and wavefront reduction of sparse matrices. *Int. J. Num. Meth. Eng.*, **23**, 239–251 (1986).
- Stadtherr, M. A. and J. A. Vegeais, Process flowsheeting on supercomputers. *ICHEME Symp. Ser.*, **92**, 67–77 (1985).
- Stadtherr, M. A. and E. S. Wood, Sparse matrix methods for equation-based chemical process flowsheeting: I. Reordering phase. *Comput. Chem. Engng*, **8**, 19–23 (1984).
- Tinney, W. F. and J. W. Walker, Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. of the IEEE*, **55**, 1801–1809 (1967).
- Vegeais, J. A. and M. A. Stadtherr, Vector processing strategies for chemical process flowsheeting. *AIChE J.*, **36**, 1687–1696 (1990).

Zitney, S. E., Sparse matrix methods for chemical process separation calculations on supercomputers. In *Proc. Supercomputing '92*, pages 414–423. IEEE Press, Los Alamitos, CA (1992).

Zitney, S. E., L. Brüll, L. Lang, and R. Zeller, Plantwide dynamic simulation on supercomputers: Modeling a Bayer distillation process. *AIChE Symp. Ser.*, **91**(304), 313–316 (1995).

Zitney, S. E. and M. A. Stadtherr, Computational experiments in equation-based chemical process flowsheeting. *Comput. Chem. Engng*, **12**, 1171–1186 (1988).

Zitney, S. E. and M. A. Stadtherr, Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Comput. Chem. Engng*, **17**, 319–338 (1993).

Figure Captions

Figure 1. (a) An occurrence matrix used as an example in the text and (b) the corresponding bipartite graph \mathcal{G} .

Figure 2. Net-column occurrence matrix and graph \mathcal{G}_N corresponding to the occurrence matrix in Figure 1(a).

Figure 3. Result of applying the RMCD ordering strategy to the occurrence matrix in Figure 1(a), with columns shown in elimination order.

Figure 4. Result of applying the MNA ordering strategy to the occurrence matrix in Figure 1(a), with columns shown in elimination order.

Figure 5. Relative performance of the row orderings based on all test matrices in Table 1 except the last six (which are not process simulation problems). Solution times have been normalized with respect to the best method on a given problem; thus an average normalized solution time of one would represent the best attainable value. See text for further discussion.

Table 1: Matrix statistics and comparison of solution times for different row orderings. See text for definition of symmetry ratio and further discussion.

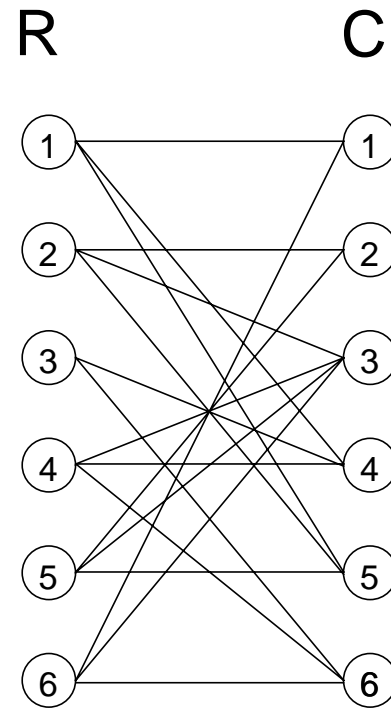
Name	Order	Nonzeros	Symmetry		Natural (s)	r-P ⁴ %	RMCD %	MNA %
			Ratio	Source				
ngc	1235	16868	.0336	SEQUEL	0.0540	245	72.1	72.0
lhr	1477	18592	.00732	SEQUEL	0.718	71.1	68.1	74.1
mult1	612	4459	.0103	SEQUEL	0.0233	61.0	75.5	72.6
mult2	714	5001	.0196	SEQUEL	0.0228	71.9	95.9	94.9
mult3	526	5363	.0329	SEQUEL	0.282	53.1	56.9	60.3
cyclo1	517	2420	.0158	SEQUEL	0.0123	77.3	80.0	83.5
beef	1197	12070	.00680	SEQUEL	0.0350	90.2	102	118
lhr2	2594	37206	.00732	SEQUEL	0.156	85.9	75.6	96.1
lhr4	4101	82682	.0152	SEQUEL	0.323	61.0	60.6	74.9
lhr7	7338	156508	.0174	SEQUEL	0.542	244	72.6	95.3
lhr10	10672	232633	.00879	SEQUEL	0.800	201	91.5	106
lhr11	10964	233741	.00820	SEQUEL	0.973	545	80.0	–
lhr14	14270	307858	.00662	SEQUEL	1.21	539	90.2	–
lhr17	17576	381975	.00151	SEQUEL	1.52	452	100.0	–
segm3_m1	1045	4963	.00524	SPEEDUP	0.0237	70.3	52.5	55.7
segm3_m2	1045	5131	.00546	SPEEDUP	0.270	72.4	84.7	176
extr1b_m1	2836	11404	.00386	SPEEDUP	0.0667	88.7	109	648
extr1b_m2	2836	11579	.00380	SPEEDUP	0.0687	92.8	118	160
extr1b_m3	2836	9227	.00780	SPEEDUP	0.593	81.8	65.3	113
hydr1c_m1	5308	23752	.00413	SPEEDUP	0.166	74.4	98.2	2450
hydr1c_m2	5308	23956	.00409	SPEEDUP	0.171	73.9	119	2450
hydr1c_m3	5308	19892	.00332	SPEEDUP	0.146	69.5	62.9	153

Table 1 (continued)

Name	Order	Nonzeros	Symmetry		Natural	r-P ⁴	RMCD	MNA
			Ratio	Source	(s)	%	%	%
west0989	989	3537	.0181	H/B	0.0215	114	78.7	94.8
west1505	1505	5445	.00110	H/B	0.0365	133	75.7	96.5
west2021	2021	7353	.00327	H/B	0.0546	155	68.2	101
traycalc	1145	20296	.117	ASPEN PLUS	0.0507	658	78.6	77.0
userupp	1269	22508	.106	ASPEN PLUS	0.0566	875	78.4	78.4
v3	1078	16937	.0918	ASPEN PLUS	0.0466	716	84.3	83.0
v10	1148	15729	.0602	ASPEN PLUS	0.0478	567	74.3	75.2
v13	834	9713	.0541	ASPEN PLUS	0.0275	345	86.0	88.0
mpex2	848	11413	.0402	ASPEN PLUS	0.0265	92.9	95.1	95.6
mpex3	2473	46503	.0617	ASPEN PLUS	0.141	489	69.4	70.8
mpex4	2478	44075	.0537	ASPEN PLUS	0.123	2830	78.9	80.1
mpmult1	2023	31894	.0486	ASPEN PLUS	0.102	642	69.7	71.1
sumb	523	4998	.0513	ASPEN PLUS	0.0144	168	91.1	92.8
uosb	523	4998	.0513	ASPEN PLUS	0.0146	163	89.8	89.5
bp_1000	822	4661	.00944	H/B	0.0239	54.2	67.8	68.4
gre1107	1107	5664	0	H/B	0.0868	78.5	101.2	56.1
mahindas	1258	7862	.0166	H/B	0.0559	42.9	38.1	41.4
lns3937	3937	25407	.850	H/B	2.92	87.7	7.92	43.1
sherman5	3312	20793	.739	H/B	0.958	111	13.8	15.7
gemat11	4929	33185	.00133	H/B	0.333	95.5	67.9	153

	1	2	3	4	5	6
1	×			×	×	
2		×	×		×	
3				×		×
4			×	×		×
5		×	×		×	
6	×		×			×

(a)

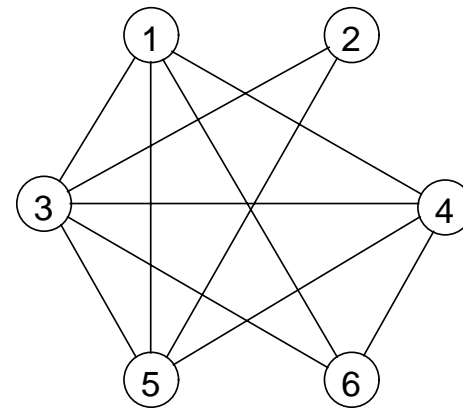


(b)

Figure 1: (a) An occurrence matrix used as an example in the text and (b) the corresponding bipartite graph \mathcal{G} .

		Column					
		1	2	3	4	5	6
Net	1			×	×	×	×
	2			×		×	
	3	×	×		×	×	×
	4	×		×		×	×
	5	×	×	×	×		
	6	×		×	×		

(a)



(b)

Figure 2: Net-column occurrence matrix and graph \mathcal{G}_N corresponding to the occurrence matrix in Figure 1(a).

	1	4	6	3	2	5
1	×	×				×
6	×		×	×		
3		×	×			
4		×	×	×		
2				×	×	×
5				×	×	×

Figure 3: Result of applying the RMCD ordering strategy to the occurrence matrix in Figure 1(a), with columns shown in elimination order.

	2	5	1	3	4	6
2	×	×		×		
5	×	×		×		
1		×	×		×	
6			×	×		×
4				×	×	×
3					×	×

Figure 4: Result of applying the MNA ordering strategy to the occurrence matrix in Figure 1(a), with columns shown in elimination order.

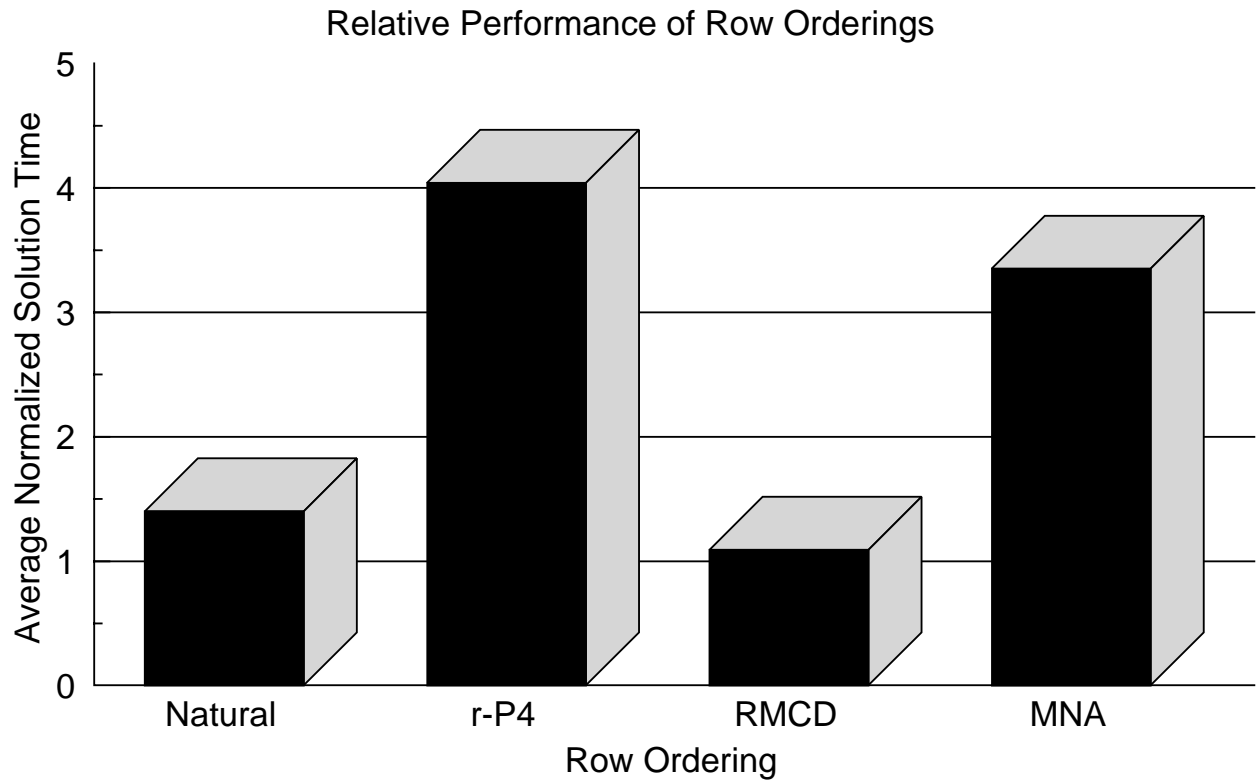


Figure 5: Relative performance of the row orderings based on all test matrices in Table 1 except the last six (which are not process simulation problems). Solution times have been normalized with respect to the best method on a given problem; thus an average normalized solution time of one would represent the best attainable value. See text for further discussion.