Finding Good Column Orderings for Sparse QR Factorization *

Pinar Heggernes
University of Bergen
Department of Informatics
N-5020 Bergen, Norway
e-mail: *pinar@ii.uib.no*

Pontus Matstoms
Linköping University
Department of Mathematics
S-581 83 Linköping, Sweden
e-mail: *pomat@mai.liu.se*

Abstract

For sparse QR factorization, finding a good column ordering of the matrix to be factorized, is essential. Both the amount of fill in the resulting factors, and the number of floating-point operations required by the factorization, are highly dependent on this ordering. A suitable column ordering of the matrix A is usually obtained by minimum degree analysis on $A^T A$. The objective of this analysis is to produce low fill in the resulting triangular factor R. We observe that the efficiency of sparse QR factorization is also dependent on other criteria, like the size and the structure of intermediate fill, and the size and the structure of the frontal matrices for the multifrontal method, in addition to the amount of fill in R. An important part of this information is lost when $A^T A$ is formed. However, the structural information from A is important to consider in order to find good column orderings. We show how a suitable equivalent reordering of an initial fill-reducing ordering can decrease the number of operations required by multifrontal QR factorization, without increasing the number of nonzeros in R. Heuristics that produce good equivalent reorderings are proposed and explained. We also propose tie-breaking strategies for the minimum degree algorithm, which produce orderings that require considerably less number of operations in both multifrontal and traditional Householder QR factorization, than the standard minimum degree algorithm without tie-breaking.

AMS Subject Classifications: 65F50, 65F25, 65F05, 65Y05
Key words: orthogonal decomposition, sparse matrix, equivalent column ordering

1 Introduction

Least squares problems play an important role in a wide range of scientific applications. Parameter estimation in mathematical modeling is one important example. Many applications lead to sparse linear least squares problems. The coefficient matrix is then assumed to be large with relatively few nonzero entries. The introduction of multifrontal methods have made direct methods based on sparse QR factorization attractive and competitive to previously recommended alternatives, as explained by Matstoms [26]. In this paper, we consider the QR factorization,

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

of a *large* and *sparse Strong Hall* matrix $A \in \mathbf{R}^{m \times n}$ of *full column rank*, where $m \geq n$. The column ordering on A is decisive for the efficiency of the factorization. Researchers

*This research is partly supported by The Research Council of Norway.

have concentrated on finding column orderings that result in as sparse R as possible. George and Heath [12] have observed that a suitable column ordering of A can be found by symmetric fill-reducing analysis of A^TA. Efficient methods, like minimum degree and nested dissection, exist for fill-reduction in symmetric factorization. Moreover, it is shown by George and Liu [14] that minimum degree analysis of A^TA can be done directly on A, without having to compute A^TA. Efficient standard methods can thus be used to compute column orderings that normally give low fill in R.

In this paper, we focus on column orderings for sparse QR factorization, and we show, by examples, that reducing the fill in R is not enough to achieve efficient factorization. Other factors, like the size and the structure of intermediate fill, and the size and the structure of the frontal matrices for multifrontal factorization, are also important for the number of operations required by the factorization process. Even for traditional sparse Householder QR factorization, the size and the structure of intermediate fill can vary considerably among column orderings that result in the same number of nonzero elements in R. In order to consider other criteria than the amount of fill in R, studying the structure of A^TA is not enough. A lot of information about A is lost when A^TA is formed, and this information from A is important in order to decide good column orderings for sparse QR factorization. Traditionally, the reason for doing a fill-reducing ordering of A^TA directly on A, has been solely to reduce the computation time. As we can see, another important reason is that, in this way, the information from A can be used in addition to the information from A^TA, which is important, for example, when trying to minimize the amount of intermediate fill, or the size of the frontal matrices.

Following the criteria mentioned above, we first concentrate on finding column orderings that reduce the number of operations required by the multifrontal QR factorization algorithm, without increasing the number of nonzeros in R. Equivalent reorderings have this property, and they are described formally later in this paper. Given any initial ordering, presumably a fill-reducing ordering, of the matrix A, equivalent reorderings can result in better column orderings than the initial ordering of A. Numerical tests show that considerable improvements, regarding the number of operations, are achieved by good equivalent reorderings. We propose heuristics for finding good equivalent reorderings, and explain the ideas behind these heuristics. These heuristics are based on the structural information from A. Numerical results, comparing these heuristics to the initial ordering, and to the best random equivalent reordering, are presented.

As opposed to the two-step approach of a good initial ordering with a proceeding equivalent reordering, we also consider a one-step approach directly applied on the original matrix A. As mentioned above, our focus is now on minimum degree analysis of A^TA done directly on A. In the minimum degree algorithm, a vertex of minimum degree is chosen at each step of the algorithm, and ties occur when there are several vertices having the minimum degree at each step. The produced ordering is dependent on how ties are resolved. The standard minimum degree algorithm resolves ties arbitrarily. However, the structure of A is important to consider when designing good tie-breaking strategies. This paper proposes methods for tie-breaking in minimum degree, which study the structure of A to produce column orderings that are suitable for sparse QR factorization. George and Ng [16] show a connection between structural Householder QR factorization and structural LU factorization. For this purpose, it is interesting to find good orderings not only for the multifrontal algorithm, but also for the traditional Householder algorithm. We compare the orderings produced by our tie-breaking strategies with orderings resulting from standard minimum degree, regarding the number of operations required by the multifrontal and the traditional Householder QR factorization algorithms.

This paper is intended to give a solid background in QR factorization and multifrontal methods. Section 2 contains a summary of recent research in this area, along with a discussion about the importance of A compared to A^TA in sparsity and efficiency analysis. Equivalent reorderings, and the relation to graph theory are explained in Section 3, and a more specific study of equivalent reorderings and QR factorization is found in Section

4. Tie-breaking strategies for minimum degree in QR factorization are explained and discussed in Section 5. Finally, Section 6 contains concluding remarks, open problems and plans for future research.

The numerical results presented in this paper are achieved through numerical tests in MATLAB versions 4.2a and 4.2c. Throughout the paper, simple MATLAB notations are used, which the reader is assumed to be familiar with. Table 1 shows the test matrices we use for our numerical experiments. Some of the matrices are from the Harwell-Boeing collection (Duff, Grimes and Lewis [5]). The others are artificially made by concatenating two Harwell-Boeing matrices. The matrices that are marked with an asterisk do not have the assumed Strong Hall Property, explained in Section 2.

Matrix	m	n	nnz	Description
ABB313	313	176	1557	Sudan Survey data
ASH219	219	85	438	Geodesy problem
ASH331	331	104	662	Geodesy problem
ASH608	608	188	1216	Geodesy problem
ASH958	958	292	1916	Geodesy problem
WL1033	1033	320	4732	Gravity-meter observations
WL1850	1850	712	8758	Gravity-meter observations
WL1252*	1252	320	5170	WL1033 , ASH219
WL1364*	1364	320	5394	WL1033 , ASH331
WL1641*	1641	320	5948	WL1033 , ASH608
WL1991	1991	320	6648	WL1033 , ASH958
WL2808	2808	712	10674	WL1850 , ASH958

Table 1: Test matrices used throughout this paper. Number of rows is m, number of columns is n, and number of nonzero elements is nnz for each matrix.

2 Sparse QR factorization

Although the method of normal equations is probably the most common method for solving dense linear least squares problems, from a numerical and computational point of view, Householder QR factorization is clearly preferable. Only well-conditioned problems can generally be satisfactorily solved by the normal equations. The situation is, however, less clear in the sparse case, where also sparsity aspects, in particular the introduction of fill, must be taken into account. Excessive fill leads to performance degradation in terms of increased execution time and storage requirements.

Among direct methods for sparse least squares, the main alternatives are the normal equations, the augmented system method and different variants of QR factorization. Methods based on QR factorization have, during the last decade, become highly interesting and recommended for sparse least squares problems (Matstoms [26]). We should keep in mind that, until the early 80s, QR factorization was rejected for general sparse problems (Duff and Reid [6]). The main explanation for why QR factorization has become useful, is the multifrontal technique. This technique makes it possible to use Householder transformations efficiently also for sparse matrices. Givens rotations can, in a very flexible way, eliminate selective nonzero entries, and should therefore be the natural choice for sparse matrices. Householder transformations are, however, from a computational point of view more attractive. First, dense QR factorization by Givens rotations is more costly than QR by Householder transformations. Second, methods using Givens rotations rely on a suitable row ordering of the matrix. A good ordering may be difficult and expensive to find. Third, Householder transformations are more easily vectorized.

In this section, we show how the multifrontal technique, originally proposed for symmetric sparse linear systems by Duff and Reid [7], can be used for sparse QR factorization. Let $A \in \mathbf{R}^{m\times n}$ be a large and sparse matrix of full rank. The described multifrontal algorithm uses Householder transformations to compute the upper triangular factor $R \in \mathbf{R}^{n\times n}$ in

$$Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{1}$$

The orthogonal factor $Q \in \mathbf{R}^{m\times m}$ is assumed not to be stored. In the solution of least squares problems, $\min_x ||Ax - b||_2$, the vector $Q^T b$ may be required, but is assumed to be computed during the factorization. In practice, this is implemented by simply merging the right-hand side vector b as an extra column to the right of A. Orthogonal transformations of A are then implicitly applied also to b. It should be mentioned that an alternative method based on QR factorization exists, making least squares problems possible to solve without using Q. In this case, an initial solution is computed by the semi-normal equations, $R^T R x = A^T b$, and then improved by one step of iterative refinement in fixed precision. Repeated least squares problems with a fixed matrix A, but with different right-hand sides, can thus be solved without refactorization.

If A has full column rank, then the symmetric matrix $A^T A$ is positive definite and has a unique triangular Cholesky factor L, $A^T A = LL^T$. From the uniqueness of L, it is easily shown that the triangular factor R in (1), except for possible sign differences of the rows, equals the Cholesky factor L^T of $A^T A$. In particular, the two matrices R and L^T have the same nonzero pattern. This relation has some important theoretical and practical implications, that will be used throughout the paper.

The above observation suggests that the sparsity pattern of R can be predicted by symbolic Cholesky factorization of $A^T A$. A priori estimation of the fill makes it, for example, possible to store R in a static data structure. Another application is to compute structural information to be used for the subsequent numerical factorization. Sparsity analysis of $A^T A$ may, however, strongly overestimate the fill. A dense row in an otherwise sparse matrix A, for example, makes $A^T A$ structurally full, although R may be sparse. Thus, further structural analysis becomes useless. One example is the following matrix, where Q is the identity matrix, and R equals the matrix A itself:

$$A = \begin{pmatrix} x & x & x & \dots & x \\ & x & & & x \\ & & x & 0 & x \\ & & & \ddots & \vdots \\ & & & & x \end{pmatrix}. \tag{2}$$

Tighter nonzero predictions of R can be obtained by symbolic Givens rotations or by Householder transformations of A. Fill is then structurally introduced according to the following propositions:

Proposition 2.1 *(Fill in sparse Householder QR factorization)*
In the elimination of column k, each column in the remaining unreduced part of the matrix having structurally nonzero inner product with column k, takes the sparsity pattern of the union of itself with column k.

Proposition 2.2 *(Fill in sparse Givens QR factorization)*
The sparsity patterns of two rows involved in a Givens transformation are replaced by the union of their nonzero patterns.

George and Heath [12] show that Givens prediction is always at least as good as symbolic Cholesky factorization. The same is true for symbolic Householder prediction, as shown by Manneback [24]. Altogether this can be summarized as follows:

$$\text{struct(R)} \subseteq \left\{ \begin{array}{l} \text{pred(Householder on } A) \\ \text{pred(Givens on } A) \end{array} \right\} \subseteq \text{pred(Cholesky on } A^T A).$$

The matrix $A \in \mathbf{R}^{m \times n}$ and its bipartite graph have the *Strong Hall Property* (SHP) if every $m \times k$ submatrix, for $1 \leq k < n$, has at least $k+1$ nonzero rows. Coleman, Edenbrandt, and Gilbert [1] show that, if A has the SHP, then symbolic Cholesky factorization, and also symbolic Givens and Householder factorization, correctly predict the nonzero structure of R. By a reordering scheme studied by Dulmage and Mendelsohn [8], [9], [10], a general matrix can be permuted into block triangular form with diagonal blocks having the SHP. In the solution of a least squares problem where the coefficient matrix has this block triangular form, only the diagonal blocks need to be factorized. It follows that in practical applications, it can always be assumed that the considered matrix has the SHP. This is also an assumption made throughout this paper. Dulmage-Mendelsohn decomposition of the matrix given in (2), which does not have the SHP, leads to 1×1 diagonal blocks.

Let P_r and P_c be row and column permutations of A. Since $(P_r A P_c)^T (P_r A P_c) = P_c^T (A^T A) P_c$, it is clear that the row ordering P_r of A has no mathematical influence on R. It should, however, be stressed that the row ordering may indeed be important for the intermediate fill; in particular if Givens rotations are used. Intermediate fill here refers to temporary nonzero entries that are introduced, but later annihilated, by the algorithm. Such fill has no influence on the final result, but may increase both the execution time and the storage requirements. Moreover, we conclude that the column ordering P_c plays the same role for the fill in R as a symmetric permutation in sparse Cholesky factorization does. The column ordering must therefore be chosen with great care.

In [12], George and Heath utilize the above relation between QR and Cholesky factorization. They observe that the problem of finding a good column ordering for QR factorization of A can be replaced by the corresponding symmetric permutation problem for sparse Cholesky factorization of $A^T A$. Well-known and efficient heuristics, like the minimum degree algorithm, can then be used. In their proposed algorithm, George and Heath form the non-zero structure of $A^T A$, pass it to a symmetric ordering heuristic, and return to the numerical factorization step where the computed column ordering of A is used. This observation and a proposed efficient scheme for sparse QR factorization by Givens rotations, makes the paper by George and Heath an important starting point for today's algorithms. References on the way from George and Heath's paper to modern multifrontal methods include Liu [22], George and Liu [13], Lewis et al. [20], Puglisi [29] and Matstoms [26].

There are essentially three important features characterizing the multifrontal method for sparse QR factorization. First, flexibility in the elimination order of the columns is provided by the underlying elimination tree. This tree gives information about structural dependencies between the columns in A. This information is of particular interest in a parallel setting, where independent columns can be eliminated in parallel. By elimination of columns, we formally mean elimination of the nonzero subdiagonal elements in the column. Second, the elimination of columns in A is performed as a sequence of *dense* subproblems. Standard Householder transformations, implemented by efficient dense linear algebra, can thus be employed and high performance can be expected on, for example, vector architectures. Third, the multifrontal algorithm avoids a lot of intermediate fill. This effect is illustrated in Figure 1, where the total amount of memory used in each elimination step is shown for the multifrontal method and for an algorithm based on the traditional Householder technique.

In the traditional Householder algorithm, columns of A are systematically eliminated from left to right. Exactly one Householder transformation is used for each column eliminated in A. After the first k elimination steps, the partially eliminated matrix equals

$$Q_k Q_{k-1} \dots Q_1 A = \begin{pmatrix} R_{11} & R_{12} \\ 0 & A^{(k)} \end{pmatrix},$$

where $(R_{11} \quad R_{12}) \in \mathbf{R}^{k \times n}$ is the first k rows of R and $A^{(k)}$ is the matrix that remains

Figure 1: Total number of stored nonzero entries in each elimination step.

to be factorized. Now, let us assume that there are h_k nonzero entries in the first column of $A^{(k)} \in \mathbf{R}^{(m-k)\times(n-k)}$. In the next elimination step, a Householder vector $u_k \in \mathbf{R}^{m-k}$ is formed, such that the Householder transformation $I - 2u_k u_k^T$ eliminates the last $h_k - 1$ nonzero elements in the first column of $A^{(k)}$. The vector u_k is zero in all elements but the h_k nonzero positions of the leading column in $A^{(k)}$. Given the Householder vectors $u_i, i = 1, \ldots, n$, the *Householder matrix* $H = [u_1, \ldots, u_n]$ is the concatenation of these vectors. Since u_i is zero in position $1 : (i-1)$, the matrix H is lower trapezoidal.

Following the above procedure, large and sparse matrices give long and sparse Householder vectors u_i. This can, however, easily be avoided by sorting the active matrices $A^{(k)}$ by rows, in such a way that rows with leading nonzero element in the first column are permuted first in the matrix. Dense Householder vectors, and dense linear algebra, can then be used. From now on, the traditional Householder algorithm refers to this modification of the procedure given above. The submatrix with row dimension h_k in $A^{(k)}$, that actually is involved in the elimination, correspond to what we later will define as a frontal matrix.

Let us now more carefully look on the multifrontal algorithm. Given a fixed column ordering of A, it is clear that columns in general must be eliminated from left to right. Fill is otherwise introduced in already processed columns. However, sparsity typically makes sets of columns structurally independent and possible to eliminate in any order, or in parallel. If two columns, i and $j > i$, of A are structurally independent, then the elimination of column i will not touch the nonzero elements in column j, and vice versa. The two columns can thus be eliminated in an arbitrary order.

It is easily shown that the columns i and j are structurally independent if and only if $r_{ij} = 0$. A compact way of describing all column dependencies is by an *elimination tree*. It is a rooted tree with n nodes and is defined from the nonzero structure of R, such that node p is the parent of node i if and only if

$$p = \min_j \{ j > i \mid r_{ij} \neq 0 \}.$$

In the elimination tree, node i corresponds to the elimination of column i in A. Let i and $j > i$ be two nodes in the tree and let $T[j]$ denote the subtree rooted at node j. Schreiber [32] shows that $r_{ij} = 0$ if $i \notin T[j]$. It follows that the columns corresponding to nodes in disjoint subtrees are independent, and can be eliminated in any order. Note, however, that r_{ij} might be zero although $i \in T[j]$. It is clear that elimination trees of low height, implying a large extent of branching and disjoint subtrees, are desirable for parallel computations. In the multifrontal method, the matrix A is completely factorized when the traversal of the elimination tree is completed, and each node, corresponding to the elimination of a column in A, is visited. Nodes can be visited in any order, as long as child nodes are visited before their parents. This is called a *topological traversal* of the elimination tree.

Since the elimination tree is obtained by symbolic prediction of R, with possible overestimation of the fill, the tree may be higher than the tree resulting from the correct R. The tree is, however, always correct if the matrix A, as we assume, has the SHP, since the predicted R is then correct. For the matrix given in (2), the correct elimination tree is a star-like tree of height two, where all nodes, but one, have the root as their parent. The elimination tree of the R predicted by the Cholesky factorization of $A^T A$ is, however, a path of length $n-1$, where node i has node $i+1$ as its parent, for $1 \leq i < n$.

To eliminate the i^{th} column of A, a *frontal matrix* F_i is formed by a contribution A_i from the matrix A, and *update matrices* U_{c_k} from the child nodes c_k, $k = 1, \ldots, t$, of node i,

$$F_i = \begin{pmatrix} A_i \\ U_{c_1} \\ \vdots \\ U_{c_t} \end{pmatrix}. \tag{3}$$

Here, A_i is the set of rows in A with leading nonzero elements in column i. To quickly access these rows, the matrix is initially sorted by rows such that the permuted matrix can be written in the form

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_n \end{pmatrix},$$

where

$$A_k = (0, \ldots, 0, \bar{a}_{kk}, \ldots, \bar{a}_{kn}) \in \mathbf{R}^{m_k \times n}$$

represents the $m_k \geq 0$ rows with leading nonzero entry in column k. The vector $\bar{a}_{kj} \in \mathbf{R}^{m_k \times 1}$ denotes the j^{th} column in this block row. By definition, $\bar{a}_{kk}$ is structurally nonzero in all components. Notice that there may be columns k in which no rows have their leading nonzero entries. For certain values of k, m_k may therefore be zero and the corresponding block row of A empty. The update matrices U_{c_k}, $k = 1, \ldots, t$, representing block rows in the active part of the partially factorized matrix, are defined and made available by the elimination of previous columns in A. Given these matrix contributions, the new frontal matrix is formed by the row merge operation (3).

The i^{th} column of A is eliminated by the elimination of the first column in F_i,

$$Q_i^T F_i = \begin{pmatrix} r_{ii} & r_{ii_1} & \ldots & r_{ii_t} \\ 0 & & & \\ \vdots & & U_i & \\ 0 & & & \end{pmatrix}. \tag{4}$$

Then the i^{th} row of R and the update matrix U_i of node i itself are computed. As soon as the update matrices of all children of a node are computed, the method can recursively continue upwards until the root node of the tree is reached and the factorization is finished.

To facilitate the use of dense linear algebra, we notice that identically zero columns in F_i remain zero after the application of orthogonal transformations. Such columns can therefore be removed in advance, making the remaining matrix dense. In practice, this modification is implemented by the use of *local* and *global* column indices. The column indices given in (4) refer to the global column indices of A.

Instead of eliminating only the first column of F_i (*restrained* elimination), also the remaining columns can be eliminated. A complete QR factorization of the frontal matrix is then computed, where the resulting triangular factor defines the i^{th} row in R and the update contribution U_i. By such *complete* elimination, the update contributions become zero below the main diagonal (upper trapezoidal form). The update matrices are otherwise dense rectangular matrices.

A natural question is now whether complete or restrained elimination should be used. From a local point of view, it is clear that the number of operations used for a certain node is minimized if only the first column is eliminated. Then the primary task is fulfilled, and there is no local reason for further elimination. However, we will show that complete elimination significantly reduces both the operation count and the storage requirements. The latter can be seen in Figure 1, where "Multifront" corresponds to complete elimination and "Traditional Householder" to restrained elimination. It should be emphasized that the use of complete elimination is one of the most important differences between multifrontal and traditional Householder QR factorization. In fact, the multifrontal algorithm with restrained elimination is, from a computational point of view, equivalent to the traditional Householder algorithm. Thus, from now on, the multifrontal algorithm will always assume complete elimination.

The most important point is that complete elimination normally reduces the number of rows in the computed update contributions. If the the frontal matrix has m_F rows and n_F columns, then the update matrix gets $\min(m_F, n_F) - 1$ rows when complete elimination is used, otherwise $m_F - 1$ rows. The upper trapezoidal update form, obtained by complete elimination, can be utilized in the frontal factorization, such that the zero entries below the diagonal are not explicitly annihilated. This is made by row-wise permutation of the frontal matrices into block upper triangular form. Numerical experiments by Matstoms [26] show that the total operation count can be reduced to the half if the block triangular form is properly used.

The main reason for using complete elimination is to minimize the amount of intermediate fill. Following Proposition 2.1, fill in the Householder method is propagated as the union of nonzero entries in the involved columns. By always keeping the update matrices as low as possible, we also minimize the amount of propagated fill. Intermediate fill increases the size of the frontal matrices, making them more costly to store and to factorize. The amount of memory used is, however, more dependent on the storage of update matrices. The rectangular update matrices from restrained elimination require much more space than the upper trapezoidal matrices obtained by complete elimination.

The multifrontal algorithm computes the QR factorization by systematically visiting the nodes in the elimination tree. Each node is associated with a frontal matrix, formed by a contribution from the original matrix A and by update matrices of the children nodes. Factorization of the frontal matrices defines a row in R and the update matrix of the node itself. It follows that update matrices temporarily must be held in memory from the point when they are computed until the parent node is reached. To minimize the amount of temporary update storage, during the topological traversal, parent nodes should be visited as soon as possible after their first child node has been visited. This can be achieved by a *postordering* of the nodes, which orders the nodes such that node j always has node $j-1$ as one of its children. A postordering of the nodes in the elimination tree, and of the columns in A, does not affect the fill in R, as explained in the next section. Postorderings can be conveniently computed by a *depth-first search* of the elimination tree, and they are a special case of topological orderings.

A number of important modifications of the described multifrontal algorithm have been proposed. The most significant improvement is *node amalgamation*, where adjacent nodes in the elimination tree are combined into *supernodes*. Such nodes correspond to the elimination of more than one column in A. Node amalgamation can, under certain conditions, be made without increasing the amount of fill. Less overhead and reduced number of floating point operations give dramatic reductions in the total execution time. A detailed study of multifrontal QR factorization can be found in Puglisi [29] and Matstoms [27].

3 Equivalent reorderings

In this section and the next, we describe column reordering strategies for A in order to obtain better properties for multifrontal QR factorization than the original column ordering of A. This section starts with a brief explanation of our purpose and approach, followed by necessary definitions and terminology. The connection between sparse matrix factorization and graph theory is given here, along with the definition of and results on equivalent reorderings. The next section is more specificly about column reordering strategies for multifrontal QR factorization.

The QR factorization of A is usually computed after an initial fill-reducing ordering on A to get low fill in R. Minimum degree is a popular fill-reducing method which has a fast implementation and which gives good results in general. As we have seen earlier, R is essentially identical to the Cholesky factor of A^TA. As suggested first by George and Heath [12], the minimum degree analysis is based on A^TA to produce low fill in the resulting R. Thus, minimum degree and other fill-reducing heuristics based on A^TA, consider only the amount of fill in R. For QR factorization, however, the number of operations required in the factorization is not only dependent on the size of R, but also on the size of intermediate fill, and the size and the structure of the frontal matrices for the multifrontal method. Since $m \geq n$, the importance of the latter mentioned properties increases as the difference between m and n increases. Important information about these properties is lost when only A^TA is considered. For good column reorderings, the information from A should also be used in addition to A^TA. Our purpose is to reduce the number of operations required in the multifrontal factorization, without introducing any new fill in R. The column reorderings of A that preserve the size of R are referred to as equivalent reorderings, and will shortly be defined formally. Our approach will be to find, among equivalent reorderings, one that tries to minimize the number of operations in multifrontal QR factorization, studying the structure of A.

We will use graph theory to describe equivalent reorderings. The reader is assumed to be familiar with standard graph terminology, but some of the terms especially related to sparse factorization are now explained. Let $M \in \mathbf{R}^{n\times n}$ be a sparse, symmetric and positive definite matrix, and let L be its Cholesky factor, so that $M = LL^T$. The graph $G = (\{1,2,...,n\}, E) = G(M)$ is a simple graph where the vertices are numbered from 1 to n, and where each vertex corresponds to a row/column of M. For each pair of vertices i and j, the edge (i,j) is in the graph if and only if $m_{ij} \neq 0$. The *filled graph* of G is the graph $G^+ = G^+(M) = G(L + L^T)$. For triangular matrices, we will simply use $G(L)$ or $G(L^T)$ instead of the correct term $G(L + L^T)$. When M and L are clear from the context, we will only use G and G^+. The filled graph G^+ can be obtained from G without computing L, with help of the following algorithm by Parter [28], called the *elimination game*:

for $i = 1 : n$,
    add edges to make all neighbors of vertex i pairwise adjacent;
    delete vertex i from the graph;
end

The graph G^+ is obtained by adding to G all the new edges suggested by this algorithm. These edges are called *fill edges* and correspond to fill elements in L. What is done in one step of this loop, is called *eliminating* vertex i. The i^{th} *elimination graph* G_i is the resulting graph after the elimination of vertex i. Thus, $G_0 = G$, and $G_n = \emptyset$. An *elimination ordering* α on G is a renumbering of the vertices of G. We use G_α to denote the renumbered graph, and G^+_α to denote the resulting filled graph. A *perfect elimination ordering* of G is an elimination ordering which results in no fill. Rose [30] has shown that all filled graphs are chordal. Chordal graphs are exactly the class of graphs which have perfect elimination orderings. A *simplicial* vertex in a chordal graph is a vertex whose neighbors induce a clique. Thus the elimination of a simplicial vertex does not produce any fill edges. Every chordal graph that is not a complete graph, has at least two nonadjacent simplicial vertices, and chordality is a hereditary property. Accordingly, a perfect elimination ordering of a chordal graph can be found by eliminating a simplicial vertex at each step of the elimination game. These results on chordal graphs are due to Dirac [2], and Fulkerson and Gross [11].

Let us show these graph issues on an example. Figure 2 shows a symmetric matrix M and its Cholesky factor L^T. The graph $G = G(M)$ and the filled graph $G^+ = G^+(M)$ are also shown. In G, vertex 1 is simplicial, thus no fill is introduced when it is eliminated first. The elimination of vertex 2 introduces the fill edge connecting vertices 3 and 4. After the elimination of 2, the remaining graph in the elimination process is complete, and all three of the vertices 3, 4 and 5 are simplicial. Thus no further fill is introduced. The filled graph G^+ is chordal, and $\alpha = [1, 5, 4, 2, 3]$ and $\beta = [1, 2, 3, 4, 5]$ are two of its perfect elimination orderings. As defined by Rose, Tarjan and Lueker [31], a *minimal fill ordering* on a graph is an ordering such that no proper subgraph of the resulting filled graph is chordal. Note that the initial ordering of M in the example of Figure 2 is a minimal fill ordering.

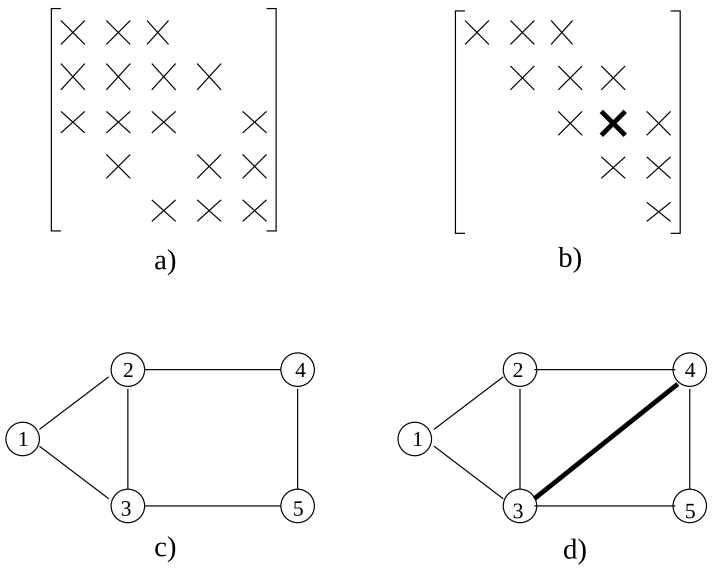

Figure 2: a) A symmetric matrix M ($M = A^T A$, where A is the nonsymmetric matrix shown in Figure 3). b) The Cholesky factor L^T of M. c) The graph $G(M)$. d) The filled graph of M, $G^+(M) = G(L)$.

An elimination ordering α on $G = G(M)$ corresponds to a symmetric permutation of the matrix M. For each ordering α, there can be found a permutation matrix P such that $G_\alpha = G(P^T M P)$. In MATLAB notation, P can be expressed as $P = I_n(:, \alpha)$, where I_n is the $n \times n$ identity matrix.

Liu [23] gives the following definition for *equivalent reorderings.*

Definition 3.1 *An elimination ordering α on G is an equivalent reordering of G if $G_\alpha^+ = G^+$, that is, the new ordering α produces a filled graph isomorphic to the original filled graph of G.*

For matrices, an equivalent reordering of M is a symmetric permutation of M that results in the same fill elements in the Cholesky factor. The Cholesky factor of the permuted matrix is also permuted, therefore the nonzero patterns of the factors of M

and P^TMP are not the same. But the two factors have the same number of nonzero elements, since their graphs are isomorphic.

Our study on equivalent reorderings is based on the following lemma.

Lemma 3.1 *Let G be a graph and $F = G^+$ be its filled graph. Let α be a perfect elimination ordering on F. Then G_α^+ is a subgraph of F_α.*

Proof: Since F is chordal, it has a perfect elimination ordering α. Elimination of the graph G_α cannot produce any fill edges that are not in F_α, otherwise these edges must also be introduced in the elimination of F_α, and α is not a perfect elimination ordering on F. To see that G_α^+ can be a proper subgraph of F_α, consider the case where G is a chordal graph and F is a complete graph as a result of a poor initial ordering on G. The perfect elimination orderings of F are thus simply all permutations of $\{1, 2, .., n\}$. Among these permutations, there is at least one ordering α which is also a perfect elimination ordering of G since G is chordal and has the same vertex set as F. Applying α on G results in zero fill, and the produced filled graph G_α^+ is a proper subgraph of F_α. □.

The following lemmas are from Liu [23].

Lemma 3.2 *If α is an equivalent reordering on G, then α is a perfect elimination ordering on the filled graph G^+.*

Lemma 3.3 *Let G be initially ordered by a minimal fill ordering. A reordering on G is equivalent if and only if it is a perfect elimination ordering of the filled graph G^+.*

As we have seen, the graph G in Figure 2 is initially ordered by a minimal fill ordering. Therefore, in this example, all perfect elimination orderings of the filled graph G^+ are equivalent reorderings on G.

Corollary 3.1 *Let $F = G^+$ be a filled graph. A reordering on F is equivalent if and only if it is a perfect elimination ordering on F.*

Proof: Since F is a filled graph, it is already numbered such that $F^+ = F$. Therefore, this is a minimal fill ordering on F, and Lemma 3.3 applies with $F^+ = F$. □

Thus for all filled graphs, the set of equivalent reorderings, and the set of perfect elimination orderings coincide. Returning to matrix notation, an equivalent reordering of M is a perfect elimination ordering of $G(L)$, whereas a perfect elimination ordering of $G(L)$, applied on M, might result in a factor with fewer nonzero elements than in L. As pointed out by Liu [23], if M is initially ordered by a fill-reducing ordering, factors resulting from equivalent reorderings on M will usually not differ considerably from the original factor L of M. Because of this, and to be able to consider a larger class of orderings, we will concentrate on the equivalent reorderings of L rather than on the equivalent reorderings of M. From now on, we use equivalent reordering to refer to one on the filled graph $G(L)$. Since L is unique for M, equivalent reorderings for M and L will be used interchangeably to mean perfect elimination orderings on $G(L)$.

Although equivalent reorderings usually result in the same filled graph, the structure of the resulting elimination tree is not necessarily the same. The filled graph is reordered, and hence the sparsity pattern of the new resulting Cholesky factor is changed, although its graph might be the same as before. Following the definition by Tarjan [33], a *topological ordering* of the elimination tree is an ordering that numbers the children nodes before their parent node. Therefore, topological orderings preserve the elimination tree. Postorderings, mentioned in Section 2, are a subset of topological orderings. Liu [21] observes that any topological ordering of the elimination tree corresponds to an equivalent reordering of the matrix M. However, as we have mentioned above, the opposite is not true. All the leaves of the elimination tree are simplicial vertices in the filled graph. But there may be more simplicial vertices in the filled graph then there are leaves in the elimination tree. Thus the number of equivalent reorderings of the graph is greater than or equal to the number of topological orderings of the elimination tree. The class of topological orderings of the elimination tree is a subset of the class of elimination tree preserving orderings, which is again a subset of equivalent reorderings of the initial matrix/graph. These relations are summarized by Liu [23].

Let us return to QR factorization (1) of A. If L is the Cholesky factor of $A^T A$, then $G(R) = G(L)$ since the nonzero structures of R and L^T are the same. In this case, equivalent reorderings of A are perfect elimination orderings of $G(R) = G(L)$. A symmetric permutation of $A^T A$ implies a column permutation on A, since $P^T(A^T A)P = (AP)^T(AP)$. Consequently, we use *equivalent column reorderings* to denote the equivalent reorderings on R.

Graph theory is also used to represent nonsymmetric matrices. A *bipartite graph* $B = (U, V, E)$ is a graph with two vertex sets U and V, and for each edge (i, j) in E, $i \in U$ and $j \in V$. The bipartite graph $B(A)$ of a nonsymmetric matrix A is defined as follows: U is the set of rows of A, V is the set of columns of A, and for each nonzero entry a_{ij}, there is an edge (i, j) in $B(A)$. We mentioned earlier that we consider Strong Hall Matrices in this paper. Strong Hall Property can also be explained by graph terminology. The bipartite graph $B(A)$ has the SHP if, for every proper subset W of the column vertex set V, the vertices in W have altogether at least $|W| + 1$ neighbors in U. For traditional Householder QR factorization, the elimination game on the bipartite graph $B = B(A)$ is defined by the following algorithm (Manneback [24]), which we call *the (Householder) bipartite elimination game*:

for $i = 1 : n$,
    let S be the set of all neighbors of column vertex i; $(S \subseteq U)$
    let T be the set of all neighbors of the vertices in S; $(T \subseteq V)$
    add edges to make all vertices in S adjacent to all vertices in T;
    delete row-vertex i and column-vertex i from the graph;
end

A similar process is described for Givens transformations by George, Liu and Ng [15]. The *filled* graph B^+ is obtained by adding to B all the edges suggested by this algorithm. The *bipartite elimination graph* B_i is the resulting graph after the elimination of column and row i. Thus, $B_0 = B$, and B_n consists of $m - n$ isolated row vertices. A connection between the set T and the neighborhood of vertex i in $G(A^T A)$ should be noted. For $i = 1$, the set T is the set of all neighbors of vertex i in $G(A^T A)$ and $G(R)$. For $i > 1$, the set T is the set of all higher numbered neighbors of i in $G^+(A^T A) = G(R)$. For $1 \leq i \leq n$, the set T is also the set of all neighbors of vertex i in the elimination graph $G_i(A^T A)$. For the bipartite elimination game algorithm to work, the matrix must have only nonzero entries in its main diagonal. Duff [3] shows that the rows of a Hall matrix can always be permuted to result in a zero-free diagonal. The *total fill* involved in the elimination process is the total of all the edges added during this algorithm. The fill edges (i, j), where $i \leq j$, correspond to fill appearing in $A^T A$ and R. The number of nonzero entries in $A^T A$ is constant for all column or row orderings of A. The number of nonzero elements in R is, however, dependent on the column ordering of A. Strictly speaking, the fill edges (i, j), where $i > j$, are called *intermediate fill* since they are later annihilated during the factorization, to give only zero entries below the main diagonal in R. However, our interpretation of intermediate fill will be slightly different in the rest of this paper. The fill which is added to $G(A^T A)$ to get $G(R)$ always appears in the upper triangle. This fill is dependent on the column ordering. Where the rest of the fill appears is dependent on the row ordering of A. Manneback mentions in [24] p. 29, that row orderings do not change the amount of total fill in the Householder bipartite elimination game. But row orderings do decide how much of the fill appears above the main diagonal, and how much below. Since row orderings are irrelevant for our work, we will onwards consider as intermediate fill, the total fill except the fill introduced between $G(A^T A)$ and $G(R)$. Figure 3 shows an example of this process. Intermediate fill is shown in dotted line style, whereas the fill in R is shown by thick lines. Consider also the symmetric matrix M in Figure 2. These two examples are related such that $M = A^T A$, where A is the matrix in Figure 3. Thus the Cholesky factor L^T of M and the QR factor R of A have the same nonzero structure. The structure of R is the upper triangular part of the filled matrix shown in Figure 3 c).

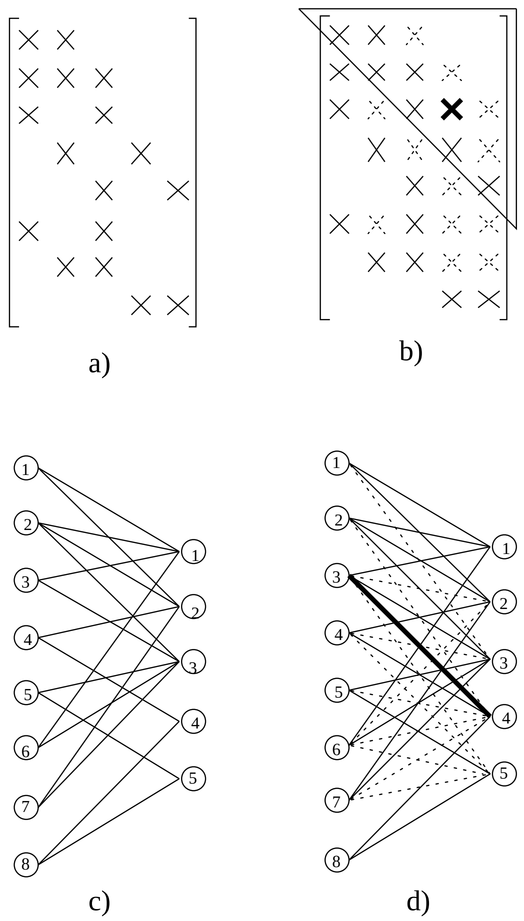

Figure 3: a) A rectangular matrix A. b) The filled matrix of A showing total fill occurring during the Householder QR factorization. The upper triangular part corresponds to R. b) The bipartite graph $B(A)$. c) The filled bipartite graph, where the dotted edges correspond to intermediate fill, and the thick edge shows fill in R.

As we have discussed, when A is initially ordered by a fill-reducing analysis based on $A^T A$, equivalent column reorderings may result in a slightly smaller R. However, the main importance of equivalent reorderings is that they can result in less number of operations in the factorization. Equivalent reorderings generally result in considerable differences in elimination tree height, amount of intermediate fill, and size and structure of frontal matrices in the factorization of A. Experimental results, presented in the next section, show that improvements, regarding the number of operations, are highly probable by an arbitrary equivalent reordering. These results also show that the best equivalent reorderings can be significantly better than the original fill-reducing ordering. Thus, with intelligent equivalent reordering strategies, the rate of improvement can be increased considerably. In the next section, we will concentrate on how equivalent reorderings can reduce the number of operations required by the multifrontal QR factorization algorithm.

4 Equivalent column reorderings for sparse QR factorization

We start this section by an example showing that the amount of intermediate fill can be changed considerably by equivalent column reorderings on the matrix to be factored. Our example matrix A is the first matrix given in Figure 3. The matrix A^TA is shown in Figure 2 a). A close study of the graph $G(A^TA)$ shows that the initial ordering of A is a fill-reducing ordering on A^TA, namely a minimum degree ordering. We have already seen that, in this example, the mentioned ordering is also a minimal fill ordering. Let us now consider an equivalent column reordering on A. That means, we want a perfect elimination ordering of the filled graph of Figure 2 d). As mentioned before, $\alpha = [1, 5, 4, 2, 3]$ is such an ordering, and G^+_α is isomorphic to G^+. Let us now reorder the columns of A to get a new matrix $B = A(:, \alpha)$, and then reorder the rows of B such that the elements on the main diagonal are all nonzero. The new matrix B is shown in Figure 4 a). In Figure 4 b) the filled matrix is shown. We see that the number of nonzeros in the upper triangular part is the same as in the factor R of A, whereas the amount of intermediate fill is considerably reduced.

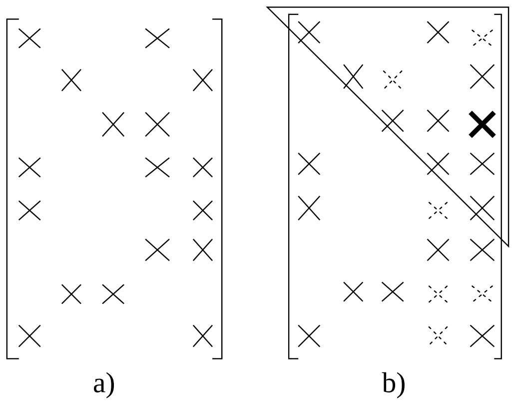

Figure 4: a) The resulting matrix B after an equivalent column reordering on matrix A of Figure 3. The rows are ordered to give only nonzeros in the main diagonal. b) The resulting filled matrix, where the upper triangular part represents the factor R' of B.

Studying the filled matrices of Figure 3 b) and Figure 4 b), some interesting observations about the occurrence of fill can be made. Since B is a result of an equivalent reordering on A, the number of nonzero entries in R' of B is always bounded above by the number of nonzeros in R. In addition, in this example, A is originally ordered by a minimal fill ordering. Therefore, the number of nonzeros in the upper triangular part of the filled matrix is constant for all equivalent reorderings of A. As can be seen from the figures, the amount of fill in upper triangle is not constant. It should be noted, however, that the number of fill edges added to $G(B^TB)$ to get $G(R')$ is constant, namely one, for every equivalent reordering on A in this example. The amount of intermediate fill can vary considerably for equivalent reorderings. What amount of this fill appears in the upper triangle as a part of B^TB, and what amount of it appears below the main diagonal, is dependent on the row ordering. We have already mentioned that intermediate fill is invariant under different row orderings, in the Householder case. The number of operations and the storage required by Householder QR factorization are proportional to the number of nonzeros in the filled matrix, which is equal to the number of edges in the filled bipartite graph $B^+(A)$. Accordingly, the number of operations is closely related to the size of the intermediate fill. Thus for traditional Householder QR factorization, the aim should be to find equivalent reorderings that reduce the amount of intermediate fill, or the number of nonzeros in the orthogonal factor Q, or the number of nonzeros in the Householder matrix H. We will shortly see that the situation is more complex for the multifrontal QR factorization algorithm.

We now leave traditional Householder QR, and continue with multifrontal QR factorization. As we have mentioned earlier, our goal is to find good orderings for the multifrontal method. Our experiments are done with the multifrontal algorithm implemented in MATLAB by Matstoms [26]. As mentioned in Section 2, the introduction of intermediate fill is quite different in the multifrontal method. If we were to define a *multifrontal bipartite elimination game*, the difference from the Householder bipartite elimination game would be that several row vertices can be removed from the graph at each step of the multifrontal game. Because of this, in the multifrontal method, the resulting filled bipartite graph $B^+(A)$ is different than in the Householder bipartite elimination game described in Section 3. The connection between the number of operations and the size of the intermediate fill is not so obvious in the multifrontal method. The results we present in this section, concern the intermediate fill and the number of operations in the multifrontal QR factorization algorithm. For our numerical experiments, we order the columns of the given matrix A by a fill-reducing analysis on A^TA. The method we have chosen is the *colmmd* function of MATLAB.

We started Section 3 by mentioning that the number of operations in multifrontal QR factorization is not only dependent on the number of nonzeros in R. Now we want to illustrate this point by an example. Figure 5 shows that various number of operations can be achieved for fixed $nnz(R)$ by equivalent reorderings. The same tendency is observed for all our test matrices, which is expressed in Observation 4.1.

Observation 4.1 *An equivalent reordering that reduces the number of nonzeros in R, does not necessarily reduce the number of operations required by the multifrontal factorization.*

As a consequence of Observation 4.1, we decide to concentrate also on other criteria when we want to find good equivalent reorderings. The example in Figure 6 shows that, for the multifrontal method, the amount of intermediate fill alone is neither enough to decide good orderings. From this we conclude that, as expected, the number of operations is dependent both on the size and the structure of the intermediate fill, and on the size and the structure of the frontal matrices, in addition to fill in R. Note, however, that for the example in Figure 6, an equivalent reordering that reduces the amount of intermediate fill will actually reduce the number of operations required in the factorization, compared to the initial fill-reducing ordering.

Observation 4.2 *An equivalent reordering that reduces the amount of intermediate fill, does not necessarily reduce the number of operations required by the multifrontal factorization.*

Observation 4.3 *The number of operations required by an equivalent reordering of A in the multifrontal method, is dependent on the number of nonzeros in R, the size and the structure of the intermediate fill, and the size and the structure of the frontal matrices resulting from the reordering.*

Our goal is to find, among equivalent column reorderings, one that reduces the number of operations in the multifrontal factorization. We want this ordering to be as close as possible to the best equivalent reordering regarding the number of operations. We concentrate on the following two criteria for the solution of this problem:

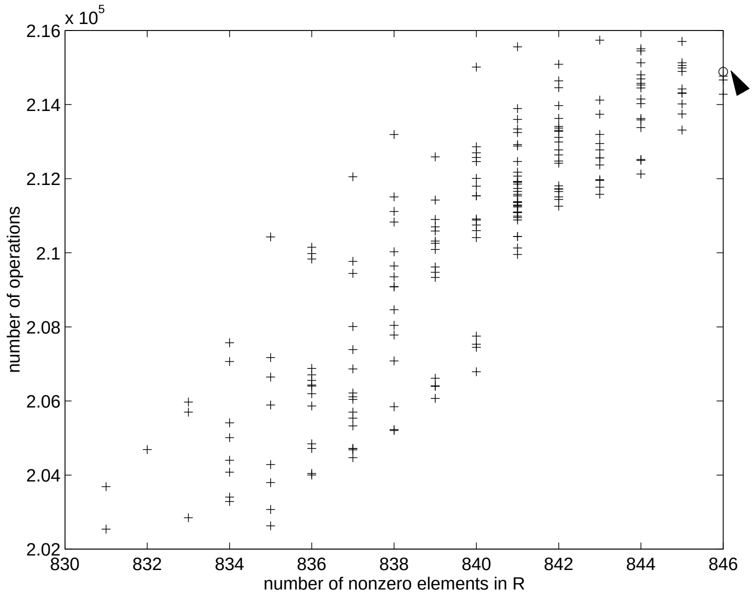

Figure 5: The columns of matrix ASH331 are ordered by the *colmmd* function of MATLAB. The chart shows the relation between the number of operations required by the multifrontal QR factorization and the size of $nnz(R)$ for 200 equivalent reorderings. Each equivalent reordering is marked with a plus. The initial *colmmd* ordering is shown by a circle.

- Size and structure of the frontal matrices.
- Size and structure of total fill.

In our heuristics, we consider perfect elimination orderings of $G(R)$. We start by examining all the simplicial vertices in $G(R)$. One of these is chosen as number 1, and is eliminated from the graph. We use the structural information from A to make this choice. At each step k, all simplicial vertices of the remaining graph are considered. Which one of these is chosen as number k depends on the heuristic, and we concentrate on the following alternatives. Choose a vertex such that:

1. Size of the k^{th} frontal matrix is minimized.
2. Number of rows with leading entry in column k of A, is minimized.
3. Number of operations to factorize the k^{th} frontal matrix is minimized.

For each of our test matrices, at least one of these heuristics produces an ordering close to the best random equivalent reordering.

Heuristic 1 The idea behind this heuristic is to locally minimize the size of the frontal matrix created at each step. The value minimized is simply the number of rows times the number of columns of the k^{th} frontal matrix. The number of rows of the frontal matrix k is the number of rows in A that have their leading entry in column k of A, in addition to the number of rows in the update matrices for column k. We approximate this number by only counting the number of rows with leading entry in column k of A. The number of columns in the frontal matrix is the number of nonzeros in row k of the resulting R.

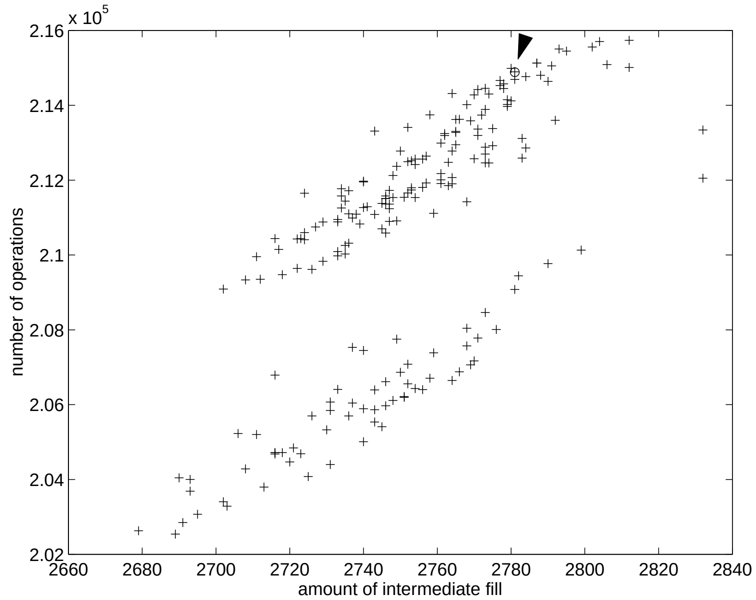

Figure 6: The columns of matrix ASH331 are ordered by the *colmmd* function of MATLAB. The chart shows the relation between the number of operations required by the multifrontal QR factorization and the size of intermediate fill for 200 equivalent reorderings. Each equivalent reordering is marked with a plus. The initial *colmmd* ordering is shown by a circle.

Heuristic 2 This heuristic minimizes the number of rows with leading entry in column k of A at each step k. The idea is to try to locally minimize the fill introduced at each step of the bipartite elimination game. The fewer nonzeros there are in the leading column, the less chance there is that fill is created as a result of these nonzeros. This is of course not completely true, because the occurrence of fill is dependent on the amount of original nonzero entries and previously created fill in this space. Clearly, minimizing fill locally does not necessarily result in a global minimization of total fill, but as a heuristic, this approach works well. Heuristic 2 has also another interpretation, it attempts to minimize locally the number of rows in the frontal matrix k.

Heuristic 3 This heuristic is similar to Heuristic 1, but minimizes the number of operations required to factorize the k^{th} frontal matrix. Since the frontal matrices are considered dense, the factorization requires approximately $ops(k) = 2n_k^2(m_k - n_k/3)$ operations for frontal matrix k, where m_k is the number of rows and n_k is the number of columns of frontal matrix k. The number m_k is approximated here as in Heuristic 1. At each step k, a column is chosen to minimize $ops(k)$.

During our work, we also considered other approaches to find good heuristics. The following strategies were tried as possible heuristics. At step k, choose a simplicial vertex of the remaining graph such that:

- Number of nonzeros in the union of column k of A and all earlier eliminated columns is minimized.
- Amount of fill introduced in the bipartite elimination graph as a result of the elimination of column k of A is minimized.

Both of these attempts gave very similar results as Heuristic 2. This is perhaps not very surprising since Heuristic 2 and these two heuristics all aim to reduce the size of total fill in the bipartite elimination. As another alternative, we formed the elimination tree, and considered the update matrices coming from the children in the elimination tree of each vertex to obtain the exact number of rows in the frontal matrix k. We then used this to minimize the number of Householder operations to factorize the k^{th} frontal matrix. But this attempt did not generally give as good results as the other heuristics.

The results from numerical tests, comparing Heuristics 1, 2, and 3 to the initial column ordering, and to the random equivalent reorderings, are given in Table 2.

Matrix	colmmd	eq. reorderings		heuristics		
		min	max	1	2	3
ABB313	430282	389354	429971	386395	375949	375436
ASH219	111798	109924	116357	109011	109011	109011
ASH331	214887	202539	215738	196292	193442	193442
ASH608	574185	524108	575705	521700	548536	548536
ASH958	1071796	920063	1072839	1062447	921211	1053683
WL1033	689017	668106	865285	782113	781870	656637
WL1252	4443569	4222301	4853924	4199075	4215067	4197257
WL1364	5074931	4952360	5442450	4982849	4928433	4963352
WL1641	14203097	14169857	16914612	14527772	13985161	14527772
WL1850	3742639	3469173	3704666	3439713	3535497	3438114
WL1991	63352154	62899480	64543723	62141096	61887380	62201316
WL2808	180635063	174184039	179673640	175131384	172559760	173969244

Table 2: Each entry in the table shows the number of flops achieved in the multifrontal factorization. The columns of each matrix are first ordered by *colmmd*. Then 200 equivalent reorderings are tested for each row of the table, and the minimum and the maximum values are given along with the results from the heuristics.

Studying the results presented in Table 2, we see that it is not easy to decide on the best heuristic method. For each matrix, at least one of the methods produces an ordering that is as good as, or even better than the best of the 200 random equivalent reorderings. However, the heuristic that gives the best result, is not the same for all matrices. Except for one matrix, at least one of the heuristics always produces a better ordering than the initial fill-reducing ordering, regarding the number of operations required by the multifrontal factorization. Except for two of the test matrices, all heuristics produce better orderings than the initial ordering. For the best results, around 10% improvement is achieved, compared to the initial ordering.

We end this section with a brief discussion on the initial fill-reducing ordering. The *colmmd* function of MATLAB, which we used as our initial fill-reducing ordering, aims to produce a minimum degree ordering of A^TA directly from A. However, this function does not produce a strict minimum degree ordering. Some approximations are done to compute a good ordering efficiently. The parameters of this function can be set from *default* to *tight* so that a strict minimum degree ordering is produced. Tests with *tight colmmd* showed that the gain from equivalent reorderings may become more restricted, although improvements are achieved also here. For some matrices, equivalent reorderings of a *default colmmd* gave better results than *tight colmmd* with or without proceeding equivalent reordering. The reader can compare the results of Table 2 to the results from *tight colmmd* presented in Table 5. This effect is in accordance with the example shown in Figure 5. An ordering that initially minimizes $nnz(R)$ need not be the best ordering for multifrontal QR factorization. Equivalent reorderings can only reduce and not increase the amount of fill in R. Observation 4.1 indicates that the best operation count and R with smallest fill need not coincide. Thus starting with a slightly more filled R than the best possible R might give better results regarding equivalent reorderings and achieving best operation count.

5 Tie-breaking strategies for the minimum degree algorithm

In the previous section, we studied the problem of reducing the cost for sparse QR factorization, by finding good *equivalent* column reorderings of the matrix A. The initial ordering was assumed to be *any* fill-reducing ordering, computed, for example, by the minimum degree algorithm. By numerical experiments, we showed both that equivalent reorderings, in general, may give a great variation in the operation count, and that our proposed heuristics, in most cases, were able to find good reorderings. The restriction to equivalent reorderings guarantees that the resulting matrix R is, at least, as sparse as the factor produced by the initial ordering. Heuristics could therefore be designed solely with respect to the operation count or intermediate fill, without fear for increased fill in R.

This section deals with the same fundamental problem: find a column ordering of A that primarily gives low fill in R, but also reduces a measure related to sparse QR factorization; for example the operation count. Our idea is to modify the minimum degree (MD) algorithm, such that *ties* are resolved with respect to intermediate fill, the operation count for Householder QR factorization, or the total length of the Householder vectors used. The last measure equals the number of nonzero entries in the Householder matrix H, defined in Section 2. Ties refer to multiple solutions of the local minimization problem, where a column of *minimum degree* should be picked. The choice of column to be eliminated next is not well-defined by the MD heuristic. By resolving ties with respect to the above criteria, the resulting triangular matrix R should be sparse, while at the same time the factorization cost is reduced.

The minimum degree algorithm (Tinney scheme 2) by Tinney and Walker [34] is a well-known and often used *symmetric* fill-reducing method. It is a greedy heuristic, and is a symmetric version of the Markowitz algorithm (Markowitz [25]) for unsymmetric matrices. Suppose that the first k columns of the symmetric matrix $M \in \mathbf{R}^{n\times n}$ have been eliminated. In the partially eliminated matrix,

$$M_k = \begin{pmatrix} R_1^{(k)} & R_2^{(k)} \\ 0 & M^{(k)} \end{pmatrix},$$

the first block row contains the first k rows of the Cholesky factor R of M. The submatrix $M^{(k)} \in \mathbf{R}^{(n-k)\times(n-k)}$ is the active part that remains to eliminate and transform into upper triangular form. In each step, the MD algorithm chooses the next column to *modify* the least number of elements in the *next* active submatrix. Let $n_i^{(k)}$ be the number of nonzero entries in the i^{th} row and column of $M^{(k)}$, then the number of modified elements in $M^{(k+1)}$ equals $(n_i^{(k)}-1)^2$. Hence, the next column is chosen such that $n_i^{(k)}$ is minimized. In terms of the elimination game, introduced in Section 3, this is equivalent to choosing a vertex in $G(M^{(k)})$ of minimal degree at each step. The graph $G(M^{(k)}) = G_k(M)$ is the resulting graph after the k^{th} step of the elimination game on $G(M)$. Since the number of operations required for eliminating column i is proportional to $[n_i^{(k)}]^2$, this criterion can be interpreted also as a local minimization of the operation count.

An alternative to the above local criterion, minimizing the number of modified elements, is the *minimum fill* criterion (Tinney scheme 3). In this case, the column to be eliminated next is chosen such that the number of *new* non-zero entries is minimized. A comparison of the minimum degree and minimum fill criteria by Duff et al. [4], shows that the difference in practice is often marginal, while the minimum fill alternative is considerably more expensive in the updates.

The minimum degree method and its implementation have received much attention during the last decades. Significant improvements in efficiency have been achieved. An excellent survey of the minimum degree algorithm and recent ideas for its efficient implementation is given by George and Liu [14].

Let us now return to sparse QR factorization, and consider how the minimum degree algorithm can be used for computing a fill-reducing column ordering of A. Following George and Heath [12], the most natural approach is to form the structure of A^TA, and pass it to symmetric minimum degree analysis. Because of symmetry, it is sufficient to compute the upper triangular part of A^TA. Since this part is contained in the structure of R, the memory space reserved for R can be used temporarily. Extra memory should not be required. This is, however, not true in implementations where computed rows of R overwrite eliminated elements of A.

A better alternative, proposed by George and Liu [14] and implemented in MATLAB by Gilbert et al. [18], is to operate directly on the matrix A. Then the nonzero structure of A^TA need not be explicitly formed, and savings in efficiency can be expected. An even more important argument, mentioned in Section 2, is that analysis on A makes it possible to include structural properties of A. Such important structural information is lost when the analysis is based on A^TA.

In Householder QR factorization, we start with the initial matrix $A^{(0)} = A$ and systematically eliminate columns, such that the triangular factor R is finally obtained. Let Q_i denote the i^{th} orthogonal transformation. The matrix obtained after k elimination steps equals

$$Q_kQ_{k-1}\dots Q_1A = \begin{pmatrix} R_{11} & R_{12} \\ 0 & A^{(k)} \end{pmatrix},$$

where $\begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \in \mathbf{R}^{k\times n}$ contains the first k rows of the final factor R. The next orthogonal transformation is given by

$$Q_{k+1} = \begin{pmatrix} I_k & 0 \\ 0 & I - 2u_ku_k^T \end{pmatrix},$$

where I_k is the identity matrix of order k and the Householder vector u_k is chosen such that the first column of $A^{(k)}$ is eliminated.

Let $B(A^{(k)}) = (U^{(k)}, V^{(k)}, E^{(k)})$ be the bipartite graph of $A^{(k)}$. The bipartite elimination game, described in Section 3, defines the successive elimination graphs $B(A^{(k)}) = B_k(A)$ for $1 \le k \le n$. In each elimination step k, and for each column vertex j in $B(A^{(k)})$, let us define two sets, $S_j^{(k)}$ and $T_j^{(k)}$. The definition of these is analogous to the definition of S and T in the bipartite elimination game: $S_j^{(k)}$ is the set of row indices in column j of $A^{(k)}$, and $T_j^{(k)}$ is the set of all columns having nonzero intersection with the rows in $S_j^{(k)}$.

Since $T_j^{(k)}$ equals the set of nonzero columns in the k^{th} row of R, it follows that local fill minimization of R is achieved, in each elimination step, by choosing a column to eliminate such that $|T_j^{(k)}|$ is minimized. This is, however, equivalent to symmetric minimum degree analysis of A^TA. To see this, remember that $T_j^{(k)}$ is the set of all neighbors of vertex j in the elimination graph $G(M^{(k)})$, where $M = A^TA$, as mentioned in Section 3. We refer to the above method as *column* minimum degree.

The column version, operating directly on A, has the same properties as the symmetric version using A^TA. Dense rows in A cause, for example, problems also in the column version. This is, at first, a bit surprising, since important structural information about A is lost when A^TA is formed. The explanation is, however, that the column version only looks on the union of column indices in the active rows, which precisely is the information used when the structure of $A^T A$ is formed.

A well-known problem in the minimum degree algorithm, is the already mentioned question of tie-breaking. In a single elimination step there are often more than one vertex of minimum degree. Although such vertices solve the local optimization problem equally well, they normally lead to different global solutions. It turns out that the amount of fill in the resulting matrix R strongly depends on the way ties are resolved. If the minimum degree vertex is chosen on random, or as the first vertex in the set of local solutions, then the tie resolution is purely determined by the initial ordering of the matrix. In an experiment in MATLAB , we generated a symmetric 200×200 sparse random matrix with density 0.02. The built-in minimum degree algorithm was used on 1000 symmetric random permutations of this matrix. Figure 7 shows the variation in nonzero count for the resulting factors.

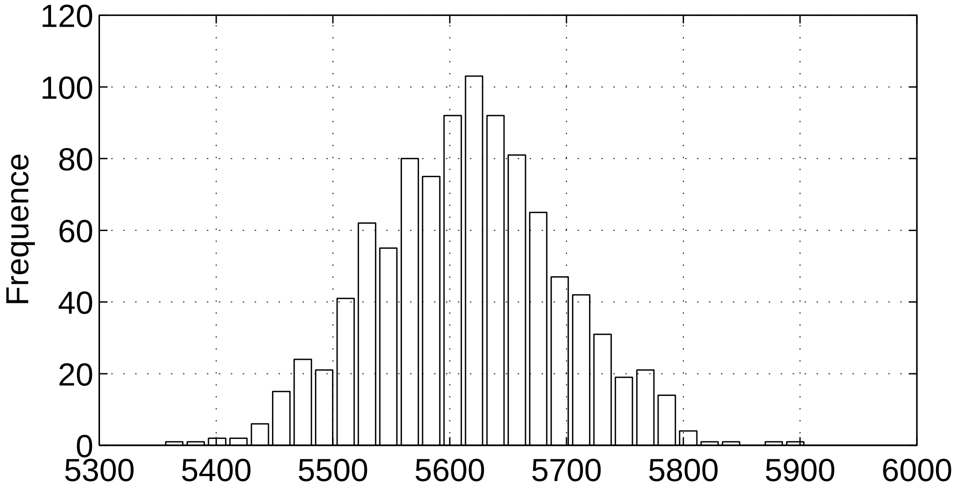

Figure 7: Distribution of $nnz(R)$ under minimum degree analysis of 1000 random permutations of a symmetric random matrix $M = \in \mathbf{R}^{200\times 200}$.

The resolution of ties is today an open problem. An efficient and practical tie-breaking strategy would improve the generally good performance of the method, and in particular make the method more stable. To reduce the effect of the initial ordering, George and Liu [14] propose the use of preorderings. Before the minimum degree algorithm is applied, the matrix is first ordered by a method less sensitive for the initial ordering. The authors notice significant improvements on a regular grid test problem when using this preordering technique. In their case, the preordering was made by the reverse Cuthill-McKee algorithm (RCM),

$$M \overset{RCM}{\longrightarrow} \tilde{M} = PMP^T \overset{MD}{\longrightarrow} \bar{M} = Q\tilde{M}Q^T.$$

The main reason for minimizing the fill, is to reduce the cost for computing and using the matrix factorization. The complexity of sparse algorithms depends, in general, on the number of nonzero entries in the result, rather than on the matrix dimension of the input matrix. The situation is, however, slightly different in sparse QR factorization. Then the operation count and the amount of memory used is rather determined by the total number of nonzero entries in the filled matrix A, as mentioned in Section 4. These elements are either from the matrix itself or intermediate fill. Therefore, the number of elements in the triangular factor as well as the intermediate fill must be minimized.

To solve this problem, we modify the minimum degree algorithm, such that the computed ordering both gives low fill in R and low intermediate fill. This is accomplished by the following general strategy:

> Use the minimum degree method to reduce fill in R, but resolve possible ties with respect to sparse QR factorization, studying the structure of A.

In our setting, tie-breaking with respect to sparse QR factorization includes either the *operation count*, the *intermediate fill*, or the *sparsity of the Householder matrix H*. These measures, all related to the *cost* for sparse QR factorization, are strongly connected and, sometimes, interchangeable.

Compared with random tie-breaking, as used in most minimum degree implementations, the proposed strategy will probably not produce more fill in R. Since, for example, fill in R and intermediate fill tend to be connected, it is rather likely that this strategy leads to less fill in R than random tie-breaking.

Following the general idea, we now describe two different tie-breaking strategies. Suppose that the minimum degree algorithm in a certain step encounters t columns (ties) $(c_1, c_2, \ldots, c_t)$ of the minimum degree d_j. The number of columns $|T_{c_i}^{(k)}|$ in $A^{(k)}$, involved in the elimination of any of these columns, is then constant and equals the common degree d_j. As mentioned before, this number also equals the number of nonzero entries in the k^{th} row of R. However, the number of rows involved in the elimination depends, in general, on the column c_i to be eliminated. Involved rows are the rows of the k^{th} frontal matrix, i.e. rows in A with leading nonzero entry in the column to be eliminated, together with rows in the update matrices of the child nodes. This is true also for the traditional Householder algorithm, but then the definition of frontal matrices is different. In this case, the k^{th} frontal matrix is given by the nonzero columns in the submatrix defined by the rows in the active matrix $A^{(k)}$ with a nonzero element in the first columns. The *sets* of involved rows and columns are denoted $S_{c_i}^{(k)}$ and $T_{c_i}^{(k)}$, respectively. In terms of these notations, we consider the two tie-breaking strategies defined in Table 3.

Strategy	Local objective function
1	$\lvert S_{c_i}^{(k)} \rvert$
2	$\lvert S_{c_i}^{(k)} \rvert \cdot \lvert T_{c_i}^{(k)} \rvert - nnz(A^{(k)}(S_{c_i}^{(k)}, T_{c_i}^{(k)}))$.

Table 3: Local objective functions in the considered two tie-breaking strategies.

Here, $nnz(A(S_{c_i}^{(k)}, T_{c_i}^{(k)}))$ denotes the number of nonzero entries in the next frontal matrix. Similar to the minimum degree heuristic and the alternative minimum fill version, the purpose in Strategy 1 is to minimize the number of operations required in the next step, while Strategy 2 minimizes the fill introduced by the next elimination step. We notice that the operation count can be minimized solely with respect to the row dimension $|S_{c_i}^{(k)}|$, since the ties all have the same degree.

In traditional Householder QR factorization, the columns of A are eliminated from left to right, such that exactly one Householder transformation is used for each column. The only difference compared to the corresponding dense algorithm, is that rows with a nonzero element in the column to be eliminated are moved together in a frontal matrix. Similar to the multifrontal method, we then get dense subproblems. This algorithm corresponds to the update procedure of the bipartite graph described above. In particular, each elimination step reduces the size of the active matrix by one row and one column.

The situation is a bit different for the multifrontal algorithm. Then a *sequence* of Householder transformations is used for the elimination of a single column in A. Instead of eliminating only the first column of each frontal matrix, complete elimination is used, such that the frontal matrices become upper trapezoidal. The size of the active matrix $A^{(k)}$ may therefore be reduced with more than one row. If the frontal matrix has h rows and d columns, then the number of rows is reduced to $\min(h, d) - 1$ rows. The number of columns is, however, always reduced by one.

We know that efficient sparse QR factorization by Householder transformations requires a method like the multifrontal scheme and *complete* factorization of the frontal matrices. The traditional Householder algorithm introduces too much intermediate fill, and can in practice therefore not be used. However, an interesting connection to sparse LU factorization (Gilbert [17], Gilbert and Ng [19]) makes the traditional Householder algorithm still interesting in real applications. The main result is presented in Theorem 5.1.

Theorem 5.1 *(George and Ng [16])*
Let $M \in \mathbf{R}^{n\times n}$ be a nonsingular square matrix with nonzero diagonal. Suppose M is factorized by Gaussian elimination with row interchanges as

$$A = P_1 L_1 P_2 L_2 \cdots P_{n-1} L_{n-1} U,$$

and define L to be the unit lower triangular matrix whose i^{th} column is the i^{th} column of L_i. Then

$$B(L + U) \subseteq B^{+}(A).$$

It follows that the nonzero structures of L and U, independently of row permutations of A, are contained in the structure of the Householder matrix H and the upper triangular factor R, respectively. Closely related to tie-breaking with respect to the operation count in QR factorization, is tie-breaking with respect to fill in H. A sparse H reduces the upper bound for the fill in LU factorization with partial pivoting. Since the row dimension $|S_{c_i}|$ of the frontal matrix associated with column c_i, is the length of the Householder vector to be used, Strategy 1 can be seen as a local method for minimizing the number of nonzero elements in H.

By the presented two tie-breaking strategies and two update procedures, we have altogether four alternative minimum degree implementations. These are summarized in Table 4.

Update procedure	Strategy 1	Strategy 2
Traditional Householder	T_1	T_2
Multifront	M_1	M_2

Table 4: The consider tie-breaking strategies and elimination procedures.

To compare the considered tie-breaking strategies, we have implemented four versions of the minimum degree algorithm as *m-files* in MATLAB. The already presented set of test matrices for the Harwell-Boeing collection was used to compare the operation count and intermediate fill. In Table 5, we compare the operation count for *multifrontal* QR factorization under the different tie-breaking strategies. We conclude that Strategy 2 always is the best. Compared with the built-in minimum degree algorithm *colmmd* (using the *tight* option), it reduces the operation count with about 20 %.

The choice of update procedure, M or T, is less important for the resulting ordering. Although only minor differences are noticed, multifrontal updates tend to be superior to traditional Householder updates. Strategy 2 under multifrontal update, M_2, is in general very stable. With one exception, it always gives a good reduction compared with the built-in random strategy.

Matrix	tight colmmd	tie-breaking			
		T_1	T_2	M_1	M_2
ABB313	300333	254656	254287	254571	254571
ASH219	66190	82799	75928	82676	82527
ASH331	188370	160485	160260	170897	170500
ASH608	532812	445708	442155	464340	429123
ASH958	793318	814998	752251	802115	709836
WL1033	732439	691041	653789	618437	619511
WL1252	4027343	3574958	3291321	3569156	3337027
WL1364	5834049	5373185	5199605	5342485	5166734
WL1641	13164543	11450903	12385206	11012698	10707118
WL1850	3589752	3258542	3261045	3255772	3274698
WL1991	41831054	43752476	40541055	40046878	40657588
WL2808	135554156	144759635	117885825	149977522	110817212

Table 5: Each entry in the table shows the number of flops achieved in the multifrontal factorization in each of the cases where the matrix is column permuted by *tight colmmd* of MATLAB and by minimum degree with our tie-breaking strategies.

Our main goal is to minimize the operation count for sparse QR factorization. This can be done by breaking ties either with respect to the operation count directly, or with respect to intermediate fill. We believe that low intermediate fill, in general, leads to low operation count. However, the connection between QR and LU also makes, as mentioned before, the sparsity of H important in its own right. Table 6 compares intermediate fill resulting from the built-in *tight colmmd* to strategy M_2, for the multifrontal algorithm. We conclude that proposed strategies, in almost all cases, reduce the intermediate fill as well as the fill in R for the multifrontal algorithm. The relative reduction of the intermediate fill is about the same as the reduction of the operation count.

Finally, we consider how the tie-breaking strategies perform when used for sparse QR factorization by the traditional Householder algorithm. The operation count for orderings computed by *default* and *tight colmmd* in MATLAB, and by the four strategies, are presented in Table 7. It is clear that Strategy 2 in general gives the best results. The elimination procedure is still less important, but in this case the T-strategies tends to beat the strategies based on multifrontal elimination.

As already mentioned, the Householder matrix H from traditional Householder plays an important roll as an upper bound for the fill in sparse LU factorization. In Table 8, we compare the number of nonzero entries in H when *tight colmmd* and strategy T_2 are used. Also in this case, the proposed strategy gives a reduction with about 15-20%.

6 Concluding remarks

In this paper, we have studied two approaches for finding good column orderings for sparse QR factorization:

1. Starting with an initial fill-reducing ordering, find a more suitable equivalent reordering, by studying the structure of A.

2. Resolve ties in minimum degree on A^TA with respect to the structure of A.

For equivalent reorderings, we have proposed heuristic methods that generally reduce the number of operations considerably. For each test matrix, at least one of the heuris-

Matrix	tight colmmd		tie-break M_2	
	intermed. fill	nonzeros in R	intermed. fill	nonzeros in R
ABB313	2693	1683	2525	1630
ASH219	1254	514	1376	544
ASH331	2594	803	2483	784
ASH608	5751	1694	5419	1630
ASH958	8966	2617	8687	2578
WL1033	8511	2607	6816	2572
WL1252	22683	3819	17821	3722
WL1364	27679	4225	23102	4152
WL1641	44435	5640	35779	5412
WL1850	23366	7504	21987	7372
WL1991	71166	9492	66441	9406
WL2808	126568	20420	107984	19533

Table 6: The tie-breaking method M_2 is here compared to *tight colmmd* regarding the amount of intermediate fill and the number of nonzeros in the resulting factor R, in the multifrontal method.

Matrix	default colmmd	tight colmmd	tie-breaking			
			T_1	T_2	M_1	M_2
ABB313	988655	862456	802100	790677	814532	808998
ASH219	261323	198175	206009	197679	221467	221083
ASH331	539540	525767	442207	441927	490492	491819
ASH608	1835662	1882747	1468051	1448897	1625482	1543707
ASH958	4438783	4260804	3576794	3499592	3934391	3476401
WL1033	3141514	3259344	2936409	2928559	2948021	2945344
WL1252	8890684	8465255	7154759	6625171	7163477	6839454
WL1364	10473678	12241484	9289946	9288404	9969582	9663951
WL1641	22501539	22873841	17356585	18077448	17074184	17074184
WL1850	20067971	20463821	14842492	14809787	14956083	15347495
WL1991	63802057	52763733	44912713	44778023	44238047	46199981
WL2808	162841784	139087318	108000550	99155210	118688482	106923782

Table 7: Tie-breaking strategies compared to *tight* and *default colmmd* regarding the number of operations in traditional Householder QR factorization.

Matrix	tight colmmd	T_2
ABB313	7252	6846
ASH219	2557	2620
ASH331	6091	5301
ASH608	14890	12422
ASH958	30618	26326
WL1033	17190	14870
WL1252	42173	36590
WL1364	53422	45067
WL1641	88516	75101
WL1850	56695	50828
WL1991	159023	146162
WL2808	268360	248886

Table 8: The tie-breaking method T_2 compared with *tight colmmd* regarding the number of nonzero entries in the Householder matrix H (traditional Householder).

tics gives, in each case, up to 10% improvement compared with the initial fill-reducing ordering.

The idea of resolving ties in minimum degree with respect to QR factorization, turns out to be very interesting. By incorporating the proposed tie-breaking strategies in the symmetric minimum degree analysis on A^TA, we reduced the cost for sparse QR factorization of A with up to 20%.

In the paper, we also focussed on the importance of studying the structure of A, in addition to A^TA, when trying to find good column orderings for QR factorization of A. This is an important problem for both multifrontal and traditional Householder QR factorization.

Some theoretical questions related to equivalent reorderings remain as open problems. We conjecture, and would like to prove, that the following two problems are NP-hard:

1. Find an equivalent reordering that minimizes the amount of total fill in the bipartite elimination game.

2. Find the best equivalent reordering in terms of the operation count for sparse QR factorization.

Our future research plans in this field include a study of the above complexity issues. In this paper, we have not concentrated on the efficient implementation of the proposed heuristics. We have rather been interested in the produced numerical results. The same is true for the tie-breaking methods. The implementation details of these heuristics are definitely subject to further research. Since we use simple MATLAB codes for our experiments, testing large matrices is time-consuming. Extended numerical comparisons, on larger matrices, require a more efficient code.

Finally, a very important and interesting problem is to find an efficient column ordering algorithm, suitable for sparse QR factorization, that works directly on A.

Acknowledgements

The authors would like to thank John R. Gilbert for useful discussions in Bergen, and for his valuable comments. We are also grateful to Bengt Aspvall, Åke Björck, and our universities, for making our trips between Bergen and Linköping possible.

References

[1] T. F. Coleman, A. Edenbrandt, and J. R. Gilbert, *Predicting fill for sparse orthogonal factorization*, J. Assoc. Comput. Mach., 33 (1986), pp. 517–532.

[2] G. A. Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.

[3] I. S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.

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

[6] I. S. Duff and J. K. Reid, *A comparison of some methods for the solution of sparse overdetermined systems of linear equations*, J. Inst. Maths. Applic., 17 (1976), pp. 267–280.

[7] ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[8] A. L. Dulmage and N. S. Mendelsohn, *Coverings of bipartite graphs*, Canad. J. Math., 10 (1958), pp. 517–534.

[9] ———, *A structure theory of bipartite graphs of finite exterior dimension*, Trans. Roy. Soc. Canada, 53, III (1959), pp. 1–13.

[10] ———, *Two algorithms for bipartite graphs*, J. Soc. Indust. Appl. Math., 11 (1963), pp. 183–194.

[11] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.

[12] J. A. George and M. T. Heath, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra and Its Applications, 34 (1980), pp. 69–83.

[13] J. A. George and J. W. H. Liu, *Householder reflections versus Givens rotations in sparse orthogonal decomposition*, Linear Algebra Appl., 88/89 (1987), pp. 223–238.

[14] ———, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[15] J. A. George, J. W. H. Liu, and E. G. Ng, *Row ordering schemes for sparse Givens transformations, I. Bipartite graph model*, Linear Algebra and Its Applications, 61 (1984), pp. 55–81.

[16] J. A. George and E. Ng, *Symbolic factorization for sparse Gaussian elimination with partial pivoting*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 877–898.

[17] J. R. Gilbert, *Private communication.*

[18] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

[19] J. R. Gilbert and E. G. Ng, *Predicting structure in nonsymmetric sparse matrix factorization*, tech. rep., Xerox Palo Alto Research Center, Palo Alto, California 94304-1314, 1992.

[20] J. G. Lewis, D. J. Pierce, and D. K. Wah, *Multifrontal Householder QR factorization*, Tech. Rep. ECA-TR-127-Revised, Boeing Computer Services, Seattle, WA, 1989.

[21] J. W. H. Liu, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. Math. Software, 12 (1986), pp. 127–148.

[22] ———, *On general row merging schemes for sparse Givens transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1190–1211.

[23] ———, *Equivalent sparse matrix reorderings by elimination tree rotations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 424–444.

[24] P. Manneback, *On some numerical methods for solving large sparse linear least squares problems*, PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1985.

[25] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.

[26] P. Matstoms, *Sparse QR factorization in MATLAB*, ACM Trans. Math. Software, 20 (1994), pp. 136–159.

[27] ———, *Sparse QR Factorization with Applications to Linear Least Squares Problems*, PhD thesis, Linköping University, Sweden, 1994.

[28] S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, 3 (1961), pp. 119–130.

[29] C. Puglisi, *QR Factorization of Large Sparse Overdetermined and Square Matrices with the Multifrontal Method in a Multiprocessing Environment*, PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 1993.

[30] D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., New York, 1972, Academic Press.

[31] D. J. Rose, R. E. Tarjan, and G. S. Leuker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Computing, 5 (1976), pp. 266–283.

[32] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[33] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, 1983.

[34] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. of the IEEE, 55 (1967), pp. 1801–1809.