ROW ELIMINATION IN SPARSE MATRICES USING ROTATIONS

Esmond Gee-ying Ng

Department of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA

CS-83-01

January 1983

ROW ELIMINATION IN SPARSE MATRICES USING ROTATIONS

by

Esmond Gee-ying Ng

A Thesis
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, 1982

ABSTRACT

One way of solving a system of linear equations $Ax=b$, where A is an m by n matrix, is to use a QR-decomposition of A (or A^T if $m<n$). In this thesis we consider the problem of computing the decomposition when A is large and sparse. The approach we use is based on row elimination using rotation matrices. The columns of A are permuted so that the triangular matrix R in the orthogonal decomposition is sparse, and the rows of A are arranged so that the cost of computing R is small. Then the rows of A are eliminated one at a time to generate R.

A graph model for studying the row and column ordering problems is proposed. The model allows us to predict the worst possible nonzero structure of R and to relate column and row orderings. Experimental results indicate that the model is a good one in the sense that the predicted structure of R is very close to the actual structure. The graph-theoretic results obtained provide us with a mechanism of constructing good row and column orderings, and of identifying good row orderings for some column orderings. Two column orderings based on dissection techniques are examined and the induced row orderings are characterized. For each of the column orderings we investigate in this thesis, numerical experiments show that there is in general a saving in execution time when the induced row ordering is used.

The methods described above assume that the matrix A^TA is sparse. There are situations in which A^TA is dense even though A is sparse. They often occur when A contains some dense rows. In those instances, it may be necessary to withhold the dense rows from the orthogonal decomposition. Algorithms for solving linear systems using the withheld rows and the orthogonal decomposition are presented.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my teacher, colleague and thesis supervisor, Professor Alan George, for generating my interest in sparse matrix computations and suggesting the topic of this thesis. His guidance, encouragement and support were invaluable, and have made this thesis a reality.

I also wish to thank the thesis committee, Professors Richard Bartels, Gregory Bennett, Ian Munro, and Robert Plemmons for their valuable comments and criticisms.

I would also like to extend my thanks to Claudia Medeiros for proofreading the final draft.

Part of the research was carried out while I was a Summer Research Intern at the Oak Ridge National Laboratory in 1981. The discussions I had with Dr. Michael Heath were helpful and stimulating.

I wish to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada through a Postgraduate Scholarship, and of the University of Waterloo.

Finally, I wish to thank the most important person in my life, Vickie, for her care, encouragement and patience. I dedicate this thesis to her.

Table of contents

CHAPTER 1

INTRODUCTION

1.1. Overview

Large sparse systems of linear equations

$$Ax = b \ ,$$

where A is an m by n matrix and b is an m-vector, arise in many engineering problems and scientific computations. Examples include underdetermined systems ($m<n$) in linear programming problems [41, 60, 67], overdetermined systems ($m>n$) in statistical computations and geodetic adjustment calculations [20, 46, 54], and square nonsingular systems ($m=n$) in structural designs [1, 5, 6]. As we describe in the next section, one way of solving a linear system involves decomposing A into either a product $Q\begin{pmatrix} R \\ O \end{pmatrix}$ where Q is an m by m orthogonal matrix and R is an n by n upper triangular matrix (for overdetermined and square systems), or a product $\begin{pmatrix} L & O \end{pmatrix}Q$ where Q is an n by n orthogonal matrix and L is an m by m lower triangular matrix (for underdetermined systems). These orthogonal decompositions will sometimes be referred to as the *QR-decomposition* and *LQ-decomposition* respectively. Note that the LQ-decomposition of a matrix A can be obtained by computing the QR-decomposition of the *transpose* of A. Thus in the following discussion we can assume that we always compute the QR-decomposition of an m by n matrix A, with $m \geq n$. We assume that A has full rank.

Orthogonal decompositions have been used extensively when A is small and dense, especially in the solution of overdetermined systems, because it is well known that such a decomposition is numerically stable and the condition of the upper triangular matrix R is the same as that of A [21, 44, 55, 74]. The storage requirement and the operation count in the

decomposition are respectively $O(mn)$ and $O(mn^2)$. See [71]. When A is large, the amount of space and the time required to compute the QR-decomposition may be prohibitively large. Fortunately, A is usually sparse if it is large; thus, sparsity should be exploited in order to minimize space and execution time.

The QR-decomposition can be obtained by reducing A column by column, or row by row. In the first approach, a sequence of m by n matrices $A^0=A$, A^1, A^2, $\cdots$, $A^n=\begin{pmatrix} R \\ O \end{pmatrix}$ is computed, where A^k is obtained from A^{k-1} by annihilating the nonzeros below the diagonal element in column k of A^{k-1} using orthogonal transformations, such as Householder transformations or rotation matrices (see [71]). That is, suppose A^{k-1} is partitioned into

$$A^{k-1} = \begin{pmatrix} R_{k-1} & S_{k-1} \\ O & \overline{A}^{k-1} \end{pmatrix},$$

where R_{k-1} is a $(k-1)$ by $(k-1)$ upper triangular matrix, S_{k-1} is a $(k-1)$ by $(n-k+1)$ matrix, and $\overline{A}^{k-1}$ is an $(m-k+1)$ by $(n-k+1)$ matrix. Then an m by m orthogonal matrix Q_k is constructed so that

$$Q_k^T A^{k-1} = \begin{pmatrix} R_k & S_k \\ O & \overline{A}^k \end{pmatrix} = A^k ,$$

where R_k is a k by k upper triangular matrix, S_k is a k by $(n-k)$ matrix, and $\overline{A}^k$ is an $(m-k)$ by $(n-k)$ matrix. See [71] for more details. The matrix $\overline{A}^{k-1}$ is usually called the *active portion* or *partially reduced matrix* in step k. For large sparse problems, such an operation usually introduces *fill-in* in $\overline{A}^k$; that is, $\overline{A}^k_{ij}$ may become nonzero even though $\overline{A}^{k-1}_{ij}$ could be zero. Here $\overline{A}^k_{ij}$ denotes the (i,j)-element of $\overline{A}^k$. Note that some of the fill-in may eventually be annihilated. As a result, the number of nonzeros in R may be much less than that in the intermediate matrix A^k. For efficient implementation of sparse QR-decomposition, a flexible scheme (a so-called *dynamic storage scheme*) must be used so that allocating space to accommodate the intermediate fill-in in A^k and the nonzeros of R, and deallocating space when a nonzero is annihilated, can be done easily. The overhead in indexing and manipulating the pointers could be large, both in

terms of space and time. In [17] the QR-decomposition computed using this approach was used to solve sparse overdetermined systems. Empirical results comparing this method with other direct methods (including the method of normal equations and a method based on Gaussian elimination) were provided and they suggested that the use of sparse QR-decomposition could be more expensive than the other direct methods. Because of the observations above, we do not consider this approach in detail.

In the row by row approach, a sequence of n by n upper triangular matrices $R^0=O$, R^1, R^2, $\cdots$, $R^m=R$ is computed, where R^k is obtained from R^{k-1} and the k-th row of A. A description of the algorithm is given in Chapter 2. The basic idea of this approach was given in [22] and a novel implementation was described in [25]. This approach is attractive for large sparse problems for the following reasons. First, only one triangular matrix and one row of A are needed during the computation. It is not necessary to store the rows of A in the main storage. They can be stored on secondary storage and read in one at a time. Second, assuming exact cancellation does not occur, one can show that any intermediate fill-in in R^k will remain nonzero in the final upper triangular matrix R. Third, it can be shown that, in the worst case, the nonzero structure of R is identical to that of the Cholesky factor of the symmetric positive definite matrix A^TA. Furthermore, it is well known that the structure of the Cholesky factor of A^TA can be determined from that of A^TA. This means that one can set up a storage scheme (a so-called *static storage scheme*) for R *before* any numerical computation begins. Storage allocation and deallocation are not necessary *during* the numerical computation and the overhead in manipulating the pointers is small. (Detailed discussions can be found in Chapter 2.) It should be noted that the m by m orthogonal matrix Q, which is large and usually dense, *is not* saved.

Recently George, Heath and Ng have compared three direct methods for solving overdetermined systems: the method of normal equations, a direct method based on Gaussian elimination, and a method based on the QR-decomposition computed using the row by row approach. They found that the method based on QR-decomposition is better, both in terms of

storage and execution time required, than the method based on Gaussian elimination. In fact, both the method of normal equations and the QR-decomposition approach are implemented using the *same* amount of space. See [26] for details.

Another important aspect of the row by row approach is that one can preserve sparsity. We pointed out that the structure of R is the same as the structure of the Cholesky factor of A^TA. It is also well known that reordering the rows and columns of A^TA may affect the sparsity of R if A^TA is sparse. It can be shown that a symmetric ordering of A^TA corresponds to a column ordering of A. Thus one can choose a column ordering so that the corresponding upper triangular matrix R is sparse.

An associated problem is whether reordering the rows of the sparse matrix A has any effect on the computation of the QR-decomposition. Experience has shown that even though reordering of the rows of A *does not* affect the structure of R, it *may* affect the *cost* (or *time*) of the computation. Relatively little is known about this problem.

In this thesis we study the problem of reordering the rows in the computation of sparse QR-decomposition using the row by row approach described in [25]. We investigate the relationship between row and column orderings. The results obtained allow us to derive "good" row orderings for some given "good" column orderings.

Note that a basic assumption we have made is that A^TA is sparse. Unfortunately, it is very easy to construct examples in which A is sparse but A^TA is dense. However, this usually occurs when some of the rows of A are relatively dense. In the last part of this thesis we derive algorithms for handling this situation.

In Section 1.2 we review how linear systems can be solved using orthogonal decompositions. In Section 1.3 we provide a brief survey of the literature on computing a sparse orthogonal decomposition. An outline of the thesis is then given in Section 1.4.

1.2. Algorithms for solving linear systems using orthogonal decompositions

In this section we review some basic algorithms for solving the system of linear equations

$$Ax = b$$

using an orthogonal decomposition of the coefficient matrix A. We assume that A is m by n and has full rank. Since the solution is computed using finite-precision arithmetic, the *computed solution* is, in general, not the same as the *exact solution*. The accuracy of the computed solution depends on the machine number system and a quantity known as the *condition number* of the coefficient matrix A [74]. The condition number of A, denoted by $cond(A)$, is a measure of the sensitivity of the computed solution to perturbations in the coefficient matrix and the right-hand side vector. The Euclidean norm will be used throughout this thesis.

Underdetermined systems ($m < n$)

The linear system $Ax=b$ is always consistent, but it has an infinite number of solutions. One of the solutions can be obtained by the following method. Suppose a QR-decomposition of A is given by

$$A = \overline{Q}\begin{pmatrix}\overline{R} & \overline{S}\end{pmatrix},$$

where $\overline{Q}$ is an m by m orthogonal matrix, $\overline{R}$ is an m by m upper triangular matrix, and $\overline{S}$ is an m by $(n-m)$ matrix. Then the linear system can be written as

$$\begin{pmatrix}\overline{R} & \overline{S}\end{pmatrix}x = \overline{Q}^T b \ .$$

Partition $x=\begin{pmatrix}x_1\\ x_2\end{pmatrix}$, where x_1 and x_2 are m- and $(n-m)$-vectors respectively. Then the linear system becomes

$$\begin{pmatrix}\overline{R} & \overline{S}\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} = \overline{R}x_1 + \overline{S}x_2 = \overline{Q}^T b \ ,$$

and a solution to the underdetermined system is given by

$$x_1 = \overline{R}^{-1}\overline{Q}^T b$$

and

$$x_2 = 0 \ .$$

In some situations, one may prefer to find the solution that has the smallest Euclidean norm, the so-called *minimal l_2-solution*. One can show that this minimal l_2-solution, which is unique, is given by

$$\overline{x} = A^T(AA^T)^{-1}b \ .$$

See [12]. Since A has full row rank, the symmetric matrix AA^T is positive definite, and has a Cholesky decomposition $AA^T = \overline{L}\,\overline{L}^T$, where $\overline{L}$ is an m by m lower triangular matrix. Thus the m-vector $y = (AA^T)^{-1}b$ can be obtained by solving two triangular systems $\overline{L}\,\overline{L}^T y = b$. However, it is well known that the *computed* AA^T may be quite different from the *actual* AA^T, because severe roundoff and/or cancellation may occur, and thus one should avoid computing AA^T explicitly. A more stable method can be derived using an LQ-decomposition of A [41, 60, 67].

Suppose an LQ-decomposition of A is given by

$$A = \begin{pmatrix} L & O \end{pmatrix} Q \ ,$$

where Q is an n by n orthogonal matrix and L is an m by m lower triangular matrix. (Note that this decomposition can be obtained from a QR-decomposition of A^T.) Then we have

$$AA^T = \begin{pmatrix} L & O \end{pmatrix} QQ^T \begin{pmatrix} L^T \\ O \end{pmatrix} = LL^T \ ,$$

and the minimal l_2-solution $\overline{x}$ is obtained by first solving two triangular systems

$$LL^T y = b$$

and then multiplying y by A^T.

Note that the accuracy of the computed solution y depends on $cond(LL^T)$ [74]. Since $AA^T=LL^T$ and $cond(AA^T)=cond(A)^2$ (see [71]), the accuracy depends on $cond(A)^2$. Thus, intuitively, the accuracy of the computed minimal l_2-solution would depend on $cond(A)^2$. However Paige has shown that, as long as A is not poorly conditioned (more precisely, $\epsilon\, cond(A)<1$, where ϵ is the machine unit roundoff error), the accuracy of the computed solution $\bar{x}$ depends essentially on $cond(A)$ [60].

Overdetermined systems ($m > n$)

In general the linear system $Ax=b$ is inconsistent; that is, it may not have a solution. In this case we usually find the unique n-vector $\bar{x}$ such that the Euclidean norm of the residual $r=b-A\bar{x}$ is minimized. This is the well-known least squares problem

$$\min_x \| Ax-b \|_2 \ .$$

Suppose a QR-decomposition of A is given by

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

where Q is an m by m orthogonal matrix and R is an n by n upper triangular matrix. Because the Euclidean norm is preserved under orthogonal transformations, we have

$$\| Ax-b \|_2 = \| Q\begin{pmatrix} R \\ O \end{pmatrix}x-b \|_2 = \| \begin{pmatrix} R \\ O \end{pmatrix}x-Q^Tb \|_2 \ .$$

Let $Q^Tb=\begin{pmatrix} c \\ d \end{pmatrix}$, where c and d are n- and $(m-n)$-vectors respectively. Then the unique least squares solution $\bar{x}$ is given by the solution to the triangular system

$$Rx = c \ .$$

This approach is due to Golub [44]. The accuracy of the computed solution depends essentially on $cond(R)$ which is the same as $cond(A)$.

In some situations, several problems having the same coefficient matrix A, but different right-hand side vectors, may have to be solved. Thus the orthogonal matrix Q must be saved so that, for each new right-hand side $\hat{b}$, $Q^T\hat{b}$ can be formed. As we have noted in the previous section, we assume that the orthogonal matrix Q is not computed explicitly, and the orthogonal transformations are not saved. Thus the solution scheme above cannot be used in those instances. An alternative is to use the normal equations

$$A^TA\hat{x} = A^T\hat{b} \quad .$$

We could compute the least squares solution by computing the Cholesky decomposition of A^TA and using the Cholesky factor to solve for $\hat{x}$. As in the case of underdetermined systems, it may be undesirable to compute A^TA explicitly because severe roundoff and/or cancellation may occur. However, it is not necessary to form A^TA when we have a QR-decomposition of A since

$$A^TA = \begin{Bmatrix} R^T & O \end{Bmatrix} Q^TQ \begin{Bmatrix} R \\ O \end{Bmatrix} = R^TR \quad .$$

Hence the system of normal equations can be written as

$$R^TR\hat{x} = A^T\hat{b} \quad .$$

Note that the orthogonal matrix Q is only needed in the computation of R; it is not needed in solving the linear system.

It should be pointed out that, in the second scheme, the accuracy of the computed solution depends on $cond(R^TR) = cond(A^TA)$. That is, the accuracy depends on $cond(A)^2$ instead of $cond(A)$. Another interesting note about the second approach is that it can be adopted to handle certain partitioned least squares problems very easily, as we discuss in Chapter 6.

Square systems ($m = n$)

The linear system $Ax=b$, where A is square and nonsingular, may be treated like an overdetermined system. That is, if a QR-decomposition of A is given by

$$A = QR \ ,$$

where Q is an n by n orthogonal matrix and R is an n by n upper triangular matrix, then the unique solution is given by the solution to the triangular system

$$Rx = Q^T b \ .$$

1.3. Survey of previous work on sparse orthogonal decomposition

In this section we provide a brief survey of work that has been done on the orthogonal decomposition of sparse matrices. An excellent survey can be found in [9].

Chen and Tewarson were among the first ones to study sparse orthogonal decomposition [11, 73]. They were interested in generating an m by n matrix Q from a sparse m by n matrix A so that the columns of Q are orthonormal. Their approach was to compute the columns of Q from the columns of A using the Gram-Schmidt process, and the columns of A were permuted so that the matrix Q was sparse. Computation using Householder transformations was also considered.

Duff considered the problem of computing a QR-decomposition of a sparse m by n matrix using rotation matrices [15]. He compared several strategies of choosing the pivot element and reordering the rows in the active portion so that the amount of intermediate fill-in could be minimized. Based on numerical experiments, Duff suggested that the pivot column should be chosen so that it had the minimum number of nonzeros in the active portion. Then, a row ordering was chosen so that the fill-in was minimized when off-diagonal nonzeros of the pivot column were annihilated. It was assumed that the annihilations were carried out column by column.

Kaufman recently considered the problem of applying Householder transformations to the columns of a sparse matrix [53]. She rearranged the computation so that the sparsity of the coefficient matrix could be exploited and the cost of the computation was small. The Householder transformations were stored explicitly. The problems of row and column orderings

were not considered.

In the papers we have considered so far, the computations were assumed to be carried out column by column. However, there is no reason why the QR-decomposition of a matrix cannot be obtained by applying rotation matrices to its rows. This approach was proposed by Gentleman for sparse matrices [22]. He suggested that the upper trapezoidal form should be "built up gradually" from the rows of the given sparse matrix. Several strategies for ordering the rows so that the amount of fill-in was small were described. Gentleman also pointed out that this approach would be attractive for very large matrices, since the rows could be stored on secondary storage and they were read in one at a time. This approach is useful because the matrix A is obtained one row at a time in many applications.

Gill and Murray also considered the elimination of rows using rotation matrices [42]. They proposed a new way of carrying out the QR-decomposition where only a sequence of sparse vectors was stored. The orthogonal matrix and the upper triangular matrix were represented in product forms. The choice of row and column orderings was ignored.

As we have mentioned in Section 1.1, George and Heath described an efficient algorithm for computing the QR-decomposition of a sparse matrix in [25], based on the idea proposed in [22]. Column ordering was chosen to reduce the number of nonzeros in the upper trapezoidal form. Their algorithm is attractive for large sparse problems because a storage scheme can be set up for the upper triangular matrix *before* the numerical computation begins and the sparse coefficient matrix can be stored on secondary storage. The orthogonal matrix was not saved. Even though some experiments were provided to illustrate the effect of row ordering on the computation time, the problem of finding a "good" row ordering problem was not investigated in detail.

George, Heath and Plemmons have implemented the algorithm of [25] for the case when the upper trapezoidal form is too large to be stored in main storage. See [27] for details.

Finally, Zlatev recently compared two pivotal strategies in sparse orthogonal decomposition using rotation matrices [77]. The paper is similar to [15] in nature. One of the pivotal strategies was the one recommended in [15], and the other one was based on some ideas proposed in [22]. The ordering decisions to be made in both strategies depended on the structure of the active portion of the matrix and the orthogonal transformation was carried out column by column.

1.4. Outline of thesis

The purpose of this thesis is to study the problem of ordering the rows in the QR-decomposition of a sparse matrix using row elimination, and to derive algorithms for handling dense rows in the solution of the m by n linear system $Ax=b$ (or dense columns in the case of underdetermined systems). In Chapter 2 we review the algorithm of George and Heath for computing the QR-decomposition of a sparse matrix and some basic results are presented. We show why the computation can be carried out using a static storage scheme. The effects of reordering the rows and columns of the coefficient matrix A are examined. A storage scheme suitable for row elimination is then described.

Chapter 3 is devoted to developing a graph model that provides us with a systematic way of studying the row ordering problem. The model can be used to predict the worst possible nonzero structure of the upper triangular matrix R obtained in the QR-decomposition. The decomposition process can also be expressed as a sequence of simple operations on graphs. Graph-theoretic results relating row and column orderings in sparse QR-decomposition using row elimination are presented. These results are important, since they provide us with some insight into how to construct good row and column orderings, and in some cases they allow us to identify good row orderings for a given column ordering. The matrix version of some of the graph-theoretic results appears in [38]. A model problem that we use to compare the quality of the proposed row orderings is then introduced. We prove that our graph model predicts correctly the nonzero structure of the upper triangular matrix R for the model problem. In the last section of

Chapter 3, we consider a restricted class of problems for which the storage requirement and the operation count can be derived if the column and row orderings proposed in Chapters 4 and 5 are used.

Based on the graph-theoretic results obtained in Chapter 3, we propose in Chapter 4 a column ordering (width-2 nested dissection ordering), which is similar to the well-known nested dissection ordering [23, 28]. This ordering tends to minimize the number of nonzeros in the upper triangular matrix R. An interesting and important result is that the width-2 nested dissection ordering induces simultaneously a row ordering which reduces the cost of computing the QR-decomposition. We show that the number of nonzeros in the final upper triangular matrix R is optimal (in the order of magnitude sense) for a model problem, and we derive the cost of computing R. We also describe how a width-2 nested dissection ordering can be generated automatically for general sparse problems. Numerical experiments are provided to demonstrate the effectiveness of the induced row ordering.

Even though width-2 nested dissection orderings are effective for most problems, the space requirements and execution times for some general sparse problems can be large. In Chapter 5 we show how a good row ordering can be characterized in a width-1 nested dissection column ordering, which is a better column ordering (in terms of storage requirements) than a width-2 nested dissection column ordering. (This is especially true in general sparse problems, assuming width-1 and width-2 nested dissection orderings do exist.) We show that, for the model problem, the complexities of width-1 and width-2 nested dissection algorithms have the same order of magnitude. We provide numerical experiments which confirm that in general width-1 nested dissection orderings are better than width-2 nested dissection orderings (in terms of storage requirements and execution times). For some general sparse problems, the improvements are large.

It is well known that the minimum degree algorithm [33, 34, 64] is a very effective (heuristic) algorithm for reducing fill-in in the Cholesky decomposition of large sparse positive

definite matrices, even though very few results about its behavior have been obtained. Since the structure of the upper triangular matrix R depends on the structure of the symmetric matrix A^TA and on the choice of the symmetric ordering, one can use the minimum degree algorithm to obtain a row and column ordering for A^TA. This provides a column ordering for the QR-decomposition of sparse matrices. At the end of Chapter 5, we present some empirical results which suggest that a minimum degree ordering is also a width-1 nested dissection ordering. Thus a good row ordering can be characterized in a manner similar to that in width-1 nested dissection. Numerical experiments are provided to show that the time required to compute the QR-decomposition is smaller when the proposed row ordering is used.

Throughout the discussion we assume that the matrix A^TA is sparse if A is sparse. This is usually true. However there are examples in which A^TA is dense even though A is sparse, and in these cases the upper triangular matrix R will usually be dense. This often occurs because a few of the rows of A are relatively dense. One way to handle this situation is to *withhold* the dense rows and compute the QR-decomposition of the *remaining* sparse portion; then the sparse upper triangular matrix and the withheld dense rows are used to solve the given linear system. We refer to the technique used in the solution process as the *updating* technique. In Chapter 6 we derive some updating algorithms for solving the three types of linear systems. Some of these algorithms appear to be new.

All numerical experiments were carried out on an IBM 4341 and single precision floating-point arithmetic was used. The programs were written in ANSI FORTRAN and compiled using the IBM Extended Optimizing FORTRAN Compiler. Times were reported in seconds. Some of the routines were taken from SPARSPAK, a FORTRAN package for solving sparse positive definite systems [30, 37].

The analyses given in Chapters 4 and 5 were done using MAPLE, an algebraic manipulation system developed at the University of Waterloo [19].

CHAPTER 2

ROW ELIMINATION USING ROTATIONS

In Chapter 1 we saw that systems of linear equations can be solved easily and stably using an orthogonal decomposition of the coefficient matrix. However this approach will not be attractive for large sparse problems unless efficient ways are available for computing such a decomposition. In this chapter we describe an efficient algorithm for reducing a sparse matrix to upper trapezoidal form using orthogonal transformations. We also look at the effects of permuting the columns and rows of the sparse matrix on such an orthogonal decomposition.

2.1. Basic algorithm for row elimination using rotations

In this section we consider the reduction of a sparse m by n matrix A, with $m \geq n$, to upper trapezoidal form using orthogonal transformations. We assume that A has full column rank. The reduction algorithm is due to George and Heath [25] and it uses so-called *rotation matrices* (or simply rotations) which are defined in the following lemma [43].

Lemma 2.1.1

Let $v = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ be a vector of length 2. Let $\gamma = \| v \|_2 = \sqrt{\alpha^2 + \beta^2}$. If $\gamma \neq 0$, then there exists an orthogonal matrix (or rotation matrix) $G = \begin{pmatrix} c & s \\ s & -c \end{pmatrix}$ such that $Gv = \begin{pmatrix} \gamma \\ 0 \end{pmatrix}$, where $c = \frac{\alpha}{\gamma}$ and $s = \frac{\beta}{\gamma}$.

□

Note that rotation matrices preserve the Euclidean norm since $\| Gv \|_2 = \gamma = \| v \|_2$.

Suppose x^T and y^T are two sparse row vectors of length n. Then one way to annihilate a nonzero element, say y_i, in y^T is to construct the appropriate rotation matrix G using x_i and y_i,

and to form the product $G\begin{pmatrix} x^T \\ y^T \end{pmatrix}$. We call x_i the *pivot element* and x^T the *pivot row*. Let $\overline{x}^T$ and $\overline{y}^T$ denote the *transformed* row vectors. Consider the j–th elements of $\overline{x}^T$ and $\overline{y}^T$, where $j \neq i$.

$$\begin{pmatrix} \overline{x}_j \\ \overline{y}_j \end{pmatrix} = G\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} c & s \\ s & -c \end{pmatrix}\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} cx_j + sy_j \\ sx_j - cy_j \end{pmatrix} .$$

There are two possibilities:

(1) $x_i \neq 0$

Now $c = \dfrac{x_i}{\sqrt{x_i^2 + y_i^2}} \neq 0$ and $s = \dfrac{y_i}{\sqrt{x_i^2 + y_i^2}} \neq 0$. Thus, assuming exact cancellation does not occur, *both* $\overline{x}_j$ and $\overline{y}_j$ are nonzero if either x_j or y_j (or both) is nonzero.

(2) $x_i = 0$

Now $c = 0$ and $s = \dfrac{y_i}{\sqrt{y_i^2}} = 1$. Thus $\overline{x}_j = y_j$ and $\overline{y}_j = x_j$. That is, the two row vectors are interchanged. In particular, if $x_k = 0$, for $1 \leq k \leq n$, then $\overline{x} = \overline{y}$ and $\overline{y} = 0$.

For future reference, we summarize these observations in the following lemma.

Lemma 2.1.2

Let x^T and y^T be two sparse row vectors of length n. Suppose a rotation matrix is constructed to operate on x^T and y^T so as to annihilate a nonzero element in y^T. Let $\overline{x}^T$ and $\overline{y}^T$ denote the transformed vectors.

(1) If the pivot element is nonzero, then the structure of the *remaining* parts of $\overline{x}^T$ and $\overline{y}^T$ is the *union* of those of x^T and y^T.

(2) If the pivot element is zero, then $\overline{x}^T = y^T$ and $\overline{y}^T = x^T$.

□

The first case is illustrated by an example in Figure 2.1.1, where the rotation matrix has been chosen to annihilate y_1.

$$\begin{pmatrix} c & s \\ s & -c \end{pmatrix}\begin{pmatrix} x^T \\ y^T \end{pmatrix} = \begin{pmatrix} c & s \\ s & -c \end{pmatrix}\begin{pmatrix} \times & 0 & \times & \times & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \\ \times & 0 & 0 & \times & 0 & \times & 0 & 0 & 0 & \times & 0 & \times \end{pmatrix}$$

$$= \begin{pmatrix} \times & 0 & \times & \times & 0 & \times & 0 & 0 & \times & 0 & \times & 0 & \times \\ 0 & 0 & \times & \times & 0 & \times & 0 & 0 & \times & 0 & \times & 0 & \times \end{pmatrix} = \begin{pmatrix} \bar{x}^T \\ \bar{y}^T \end{pmatrix}$$

Figure 2.1.1 Example showing fill-in that occurs when two sparse rows are transformed by a rotation.

Note that when the pivot element is zero, $\bar{x}^T$ and $\bar{y}^T$ can be obtained without performing any arithmetic operations. But if the pivot element is nonzero, the number of multiplicative operations required to compute $\bar{x}^T$ and $\bar{y}^T$ is then proportional to the number of nonzeros in the *transformed pivot row.* Thus a simple way of measuring the *arithmetic cost* in the latter case is to count the number of nonzeros in the transformed pivot row. For example, in Figure 2.1.1, the cost is therefore 7.

Now consider using such rotation matrices to reduce a sparse m by n $(m \geq n)$ matrix A to upper trapezoidal form. Our approach is that the computation begins with an "empty" n by n upper triangular matrix R^0 (that is, $R^0 = O$). Then a sequence of n by n upper triangular matrices, $R^1, R^2, \cdots, R^m = R$, is computed, where R^k is obtained from R^{k-1} by *rotating in* the k*-th* row of A.

Suppose that the first $(k-1)$ rows of A have been processed (or eliminated) to generate R^{k-1}. Denote the k*-th* row of A by a^k and its elements by a^k_j, $1 \leq j \leq n$. The (i,j)-element of R^{k-1} is denoted by R^{k-1}_{ij}. The following algorithm rotates a^k into R^{k-1}.

Algorithm 2.1.1 (row elimination using rotations)

(1) Let $\eta = \min\{j \mid a_j^k \neq 0, 1 \leq j \leq n\}$. That is, a_η^k is the *first* nonzero in row a^k.

(2) Use row η of R^{k-1} as pivot row and $R_{\eta\eta}^{k-1}$ as pivot element to construct a rotation matrix (see Lemma 2.1.1) to annihilate a_η^k. By Lemma 2.1.2 this may introduce nonzeros (fill-in) in both a^k and row η of R^{k-1}. For simplicity the transformed row and the upper triangular matrix will still be denoted by a^k and R^{k-1} respectively. Note that a_η^k should now be zero.

(3) If a^k does not contain any more nonzeros, then it has been rotated into R^{k-1} (or eliminated). Otherwise go to step 1.

Note that if row η of R^{k-1} is initially empty, then eliminating a^k is the same as transferring the entire row a^k into row η of R^{k-1} (see Lemma 2.1.2).

After a^k has been eliminated, the upper triangular matrix then becomes R^k.

The lemma below follows from the fact that only the diagonal elements of R^{k-1} are used as pivot elements.

Lemma 2.1.3

If $R_{ii}^k = 0$, then $R_{ij}^k = 0$, for $j > i$.

□

An important observation is that the rows of A are eliminated *sequentially*. Exactly one row of A, say a^k, and the upper triangular matrix R^{k-1} are needed in each step. It is not necessary to store the entire matrix A in main storage at any point during the computation. The rows of A can be stored in secondary storage and read in one by one when they are needed. This approach is particularly attractive when A is large. It will be shown later in this section that for the sequence of upper triangular matrices, $R^1, R^2, \cdots, R^m$, all we need in main storage is a storage scheme to accommodate R^m.

Two examples illustrating the elimination process are given in Figures 2.1.2 and 2.1.3. Nonzero elements of R^{k-1} and a^k are denoted by $\times$, nonzeros (fill-in) introduced into R^k and a^k

due to the elimination of a^k are denoted by $+$, and all elements involved in the elimination are circled. Of course elements in a^k denoted by $\oplus$ are ultimately annihilated.

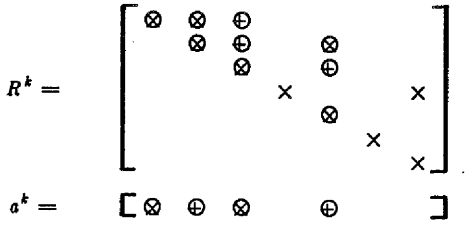


Figure 2.1.2 A sparse upper triangular matrix R^k where circled elements are involved in the elimination of a^k. Elements introduced into R^k and a^k due to the elimination of a^k are denoted by $\oplus$.

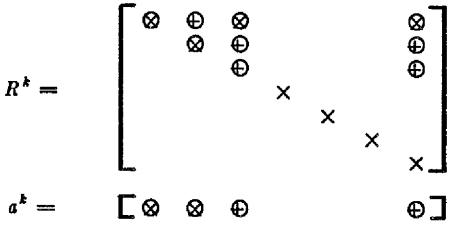


Figure 2.1.3 Another example which is similar to the one given in Figure 2.1.2.

We call the increasing sequence of row indices involved in the elimination of a^k its *elimination sequence*, which we denote by $\Xi^k=\{\xi_1^k,\xi_2^k,\cdots,\xi_{\mu_k}^k\}$. Here μ_k is called the *length* of Ξ^k. In Figure 2.1.2, $\Xi^k=\{1,2,3,5\}$, and in Figure 2.1.3, $\Xi^k=\{1,2,3\}$. Clearly $1\leq\xi_j^k\leq n$ for $1\leq j\leq\mu_k$. The following lemma is an immediate consequence of the elimination process.

Lemma 2.1.4

Let $\Xi^k=\{\xi_1^k,\xi_2^k,\cdots,\xi_{\mu_k}^k\}$ be the elimination sequence of row a^k. Then

(1) ξ_1^k is the column index of the first nonzero in a^k.

(2) ξ_{i+1}^k is the column index of the first off-diagonal nonzero in row ξ_i^k of R^k, for $1\leq i<\mu_k$.

□

Note that the elimination sequence terminates for one of two reasons:

(1) Row $\xi_{\mu_k}^k$ of R^k has no off-diagonal nonzeros, as in the example of Figure 2.1.2.

(2) Row $\xi_{\mu_k}^k$ of R^{k-1} is empty. Hence the row being eliminated can be transferred into row $\xi_{\mu_k}^k$ of R^{k-1}, as in the example of Figure 2.1.3.

In case (1), the elimination sequence is said to be *maximal*. When $m\gg n$, there will usually be more maximal elimination sequences than non-maximal ones. The following lemma relates the members of an elimination sequence Ξ^k to the nonzero structure of the upper triangular matrix R^k. It is a consequence of Lemmas 2.1.2 and 2.1.4.

Lemma 2.1.5

Let s and t be consecutive members of Ξ^k. Then

(1) $R_{st}^k\neq 0$,

(2) if $t-s>1$, then $R_{sj}^k=0$ for $s<j<t$, and

(3) if $R_{sj}^k\neq 0$, then $R_{tj}^k\neq 0$, for $j\geq t$.

□

The next result follows from Lemma 2.1.5 and the definition of maximal elimination sequences.

Lemma 2.1.6

Let $\exists^k$ be a maximal elimination sequence and $s \in \exists^k$. If $R^k_{sj} \neq 0$, $t > s$, then $t \in \exists^k$.

□

Let $\lambda^k(p)$ be the number of nonzeros in row p of R^k. Then, using Lemma 2.1.2 and the definition of cost of annihilating a nonzero, the *cost* of eliminating row a^k is therefore $\sum_{p \in \exists^k} \lambda^k(p)$. That is, the cost depends on the structure of R^k, which depends on the structures of R^{k-1} and a^k. Note that if the rotation of a row into R^{k-1} simply amounts to transferring it into R^{k-1}, the cost of such an operation, according to our definition, is also given by the number of nonzeros in the given row. We adapt this definition for simplicity, and because in most cases time proportional to the number of nonzeros in the row will be expended, even if it is not done so in performing arithmetic. In any case, the error introduced in the cost is at most $O(\min(|A|, |R|))$, and problems where such a term is significant with respect to the execution time bound are probably too small or special to be of much practical interest. Here, $|M|$ denotes the number of nonzeros in M where M is either a vector or a matrix.

So far we have only described how the reduction can be performed using row transformations. We have not discussed how we can allocate space to accommodate the fill-in that occurs during the transformation process. If the upper triangular matrices R^k are large and sparse, we want to store only the nonzeros. In order to reduce any overhead in space and time, we need some mechanism to predict where the nonzeros in R^k will be *before* the numerical computation begins, so that we can set up a storage scheme to accommodate them. The following results provide us with a solution to this problem.

Lemma 2.1.7

Assume exact cancellation does not occur. If $R_{ij}^{k-1} \neq 0$, then $R_{ij}^{k} \neq 0$.

Proof:

If $i \notin \exists^k$, then row i of R^{k-1} will not be used as pivot row in the elimination of a^k. Hence its nonzero structure will not be affected.

If $i \in \exists^k$, then either $R_{ii}^{k-1}=0$ or $R_{ii}^{k-1} \neq 0$. There is nothing to prove if $R_{ii}^{k-1}=0$ because of Lemma 2.1.3. Thus assume $R_{ii}^{k-1} \neq 0$. Then by Lemma 2.1.2, after the transformation the structure of row i of R^k is the union of those of row i of R^{k-1} and a^k. This proves the lemma.

□

Theorem 2.1.8

For $1 \leq k \leq m$, if $R_{ij}^{k} \neq 0$, then $R_{ij} \neq 0$.

Proof:

This follows from Lemma 2.1.7.

□

Hence all we need is the nonzero structure of the *final* upper triangular matrix R (i.e., R^m). If we can determine the nonzero structure of R, we can then set up a storage scheme for R and there will *always* be space to accommodate *any* fill-in in the elimination of any row.

Note that algebraically the elimination process can be expressed as the following.

$$Q_m^T Q_{m-1}^T \cdots Q_2^T Q_1^T A = \begin{pmatrix} R \\ O \end{pmatrix} .$$

Here Q_k is an m by m orthogonal matrix which is the product of the rotation matrices used in the elimination of a^k. Let

$$Q = Q_1 Q_2 \cdots Q_{m-1} Q_m .$$

Then we have

$$A = Q \begin{pmatrix} R \\ O \end{pmatrix} .$$

A useful observation is that

$$A^T A = \begin{pmatrix} R^T & O \end{pmatrix} Q^T Q \begin{pmatrix} R \\ O \end{pmatrix} = \begin{pmatrix} R^T & O \end{pmatrix} \begin{pmatrix} R \\ O \end{pmatrix} = R^T R \ .$$

Since we assume that A has full column rank, then $A^T A$ is an n by n, symmetric and positive definite matrix. It is well known that $A^T A$ can be uniquely decomposed into the product LL^T (the Cholesky decomposition), where L is an n by n lower triangular matrix with positive diagonal elements. Hence $R^T R$ is the Cholesky decomposition of $A^T A$, apart from possible sign differences in some rows of R. It is also well known that given a sparse symmetric positive definite matrix $A^T A$, the nonzero structure of its Cholesky factor R can be determined from that of $A^T A$ *before* any numerical computation begins -- a process which is usually known as *symbolic factorization* [31, 35]. Thus this provides us with a way of setting up a storage scheme for R.

There are two important points to make about this approach. First we are assuming that the matrix $A^T A$ is sparse, even though there are examples in which the matrix A is sparse but $A^T A$ is dense. We will consider this in more detail in Chapter 6. For the moment we assume that $A^T A$ is sparse. Second the structure of R determined from that of $A^T A$ using the symbolic factorization process may be pessimistic. In other words, the number of nonzeros in the *computed* upper triangular matrix R may be less than that predicted by a symbolic factorization procedure. This discrepancy is due to the fact that R is computed from A while its structure is determined from the structure of $A^T A$. More on this can be found in Chapter 3.

We conclude our discussion in this section by presenting the algorithm for reducing A to upper trapezoidal form.

Algorithm 2.1.2 (orthogonal reduction of sparse matrix)

(1) Determine the nonzero structure of A^TA.

(2) Perform a symbolic factorization on A^TA to determine the nonzero structure of R and set up an appropriate storage scheme for R.

(3) Use Algorithm 2.1.1 to process the rows of A to generate R.

2.2. Effect of column ordering

Suppose P_c is any n by n permutation matrix. Then post-multiplying the m by n matrix A by P_c effects a reordering of the *columns* of A. Denote AP_c by $\overline{A}$. Let $\overline{R}$ be the upper triangular matrix obtained by eliminating the rows of $\overline{A}$ using rotations. That is,

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

for some m by m orthogonal matrix $\overline{Q}$.

Note that $\overline{R}$ is the Cholesky factor of the symmetric positive definite matrix $\overline{A}^T\overline{A}$ since $\overline{R}^T\overline{R}=\overline{A}^T\overline{A}$. Furthermore, since

$$\overline{A}^T\overline{A} = (AP_c)^T(AP_c) = P_c^T(A^TA)P_c \quad ,$$

$\overline{A}^T\overline{A}$ is obtained simply by permuting the rows and columns of A^TA symmetrically. This symmetric permutation is identical to P_c. Note that $|\overline{A}^T\overline{A}| = |A^TA|$, *but* their nonzero structures are different. Intuitively the structure of $\overline{R}$ should depend on *both* P_c and the structure of A. In fact, it is well known that given a sparse symmetric positive definite matrix A^TA, the choice of the permutation matrix P_c can drastically affect the sparsity of $\overline{R}$ [36, 64]. This is illustrated by an example in Figure 2.2.1 in which R is completely full even though A^TA is sparse. However, after permuting the rows and columns symmetrically, the matrix $\overline{R}$ is as sparse as $\overline{A}^T\overline{A}$.

$$A^T A = \begin{pmatrix} 5 & 1 & 1 & 1 & 1 \\ 1 & 4 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 2.2361 & 0.4472 & 0.4472 & 0.4472 & 0.4472 \\ 0 & 1.9494 & -0.1026 & -0.1026 & -0.1026 \\ 0 & 0 & 1.6702 & -0.1261 & -0.1261 \\ 0 & 0 & 0 & 1.3318 & -0.1700 \\ 0 & 0 & 0 & 0 & 0.8629 \end{pmatrix}$$

$$\overline{A}^T \overline{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 & 1 \\ 0 & 0 & 0 & 4 & 1 \\ 1 & 1 & 1 & 1 & 5 \end{pmatrix}$$

$$\overline{R} = \begin{pmatrix} 1.0000 & 0 & 0 & 0 & 1.0000 \\ 0 & 1.4142 & 0 & 0 & 0.7071 \\ 0 & 0 & 1.7321 & 0 & 0.5774 \\ 0 & 0 & 0 & 2.0000 & 0.5000 \\ 0 & 0 & 0 & 0 & 1.7078 \end{pmatrix}$$

Note: $\overline{A}^T \overline{A} = P_c^T (A^T A) P_c$, where

$$P_c = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2.2.1 Examples of fill-in in the Cholesky decomposition of sparse symmetric positive definite matrices.

Hence in Algorithm 2.1.2 we want to determine, *before* eliminating the rows, a *column permutation* P_c for A (that is, a symmetric row and column permutation for A^TA) such that the upper triangular matrix $\overline{R}$ is sparse.

Yannakkis has recently proved that finding an *optimal* permutation P_c for a sparse symmetric positive definite matrix is an NP-complete problem [76] (optimal in the sense that the number of nonzeros in $\overline{R}$ is minimal). However, reliable heuristic algorithms are available for finding a permutation P_c that yields a reasonably sparse $\overline{R}$. Examples are the minimum degree algorithm [33, 34, 64] and the nested dissection algorithm [23, 28]. Thus we can apply one of these heuristic algorithms to the matrix A^TA to provide us with a good permutation P_c before we perform the row elimination. This is summarized in the following algorithm.

Algorithm 2.2.1 (orthogonal reduction of sparse matrix - with column permutation)

(1) Determine the nonzero structure of A^TA.

(2) Determine a permutation P_c for A^TA so that the Cholesky factor $\overline{R}$ of $P_c^T(A^TA)P_c$ is sparse.

(3) Perform a symbolic factorization on $P_c^T(A^TA)P_c$ to determine the nonzero structure of $\overline{R}$ and set up an appropriate storage scheme for $\overline{R}$.

(4) Use Algorithm 2.1.1 to rotate the rows of AP_c sequentially into $\overline{R}$.

2.3. Effect of row ordering

In the last section we showed that the sparsity of the upper triangular matrix $\overline{R}$ depends on the given matrix A and the choice of the column permutation matrix P_c. What about permuting the rows of A? Does it have any effect on $\overline{R}$?

Let P_r be any m by m permutation matrix. Pre-multiplying the m by n matrix AP_c by P_r corresponds to permuting the *rows* of AP_c. Denote P_rAP_c by $\hat{A}$. Suppose $\hat{A}$ has an orthogonal decomposition given by

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

where $\hat{Q}$ is an m by m orthogonal matrix and $\hat{R}$ is an n by n upper triangular matrix. Here $\hat{R}$ is the Cholesky factor of the symmetric positive definite matrix $\hat{A}^T\hat{A}$. The first observation is that $\hat{R}=\overline{R}$ since

$$\hat{R}^T\hat{R} = \hat{A}^T\hat{A} = (P_rAP_c)^T(P_rAP_c) = P_c^TA^TP_r^TP_rAP_c = P_c^TA^TAP_c = \overline{R}^T\overline{R} .$$

That is, row permutations *do not* affect the sparsity of $\overline{R}$. So what is the role of a row permutation in the orthogonal reduction of a sparse matrix?

Note that the matrix $\hat{R}$ is obtained by generating a sequence of upper triangular matrices,

$$\hat{R}^0 = O, \hat{R}^1, \hat{R}^2, \cdots, \hat{R}^{m-1}, \hat{R}^m = \hat{R} = \overline{R} ,$$

where $\hat{R}^k$ is obtained from $\hat{R}^{k-1}$ by rotating in the k–*th* row of $\hat{A}=P_rAP_c$. Thus the *structure* of $\hat{R}^k$ depends on those of $\hat{R}^{k-1}$ and the k–*th* row of $\hat{A}$. In other words, the structure of $\hat{R}^k$ depends on those of the first k rows of $\hat{A}$. Note that the first k rows will be different for different choice of P_r.

Recall from Section 2.1 that the cost of eliminating row k of $\hat{A}$ depends on the elimination sequence and the structure of $\hat{R}^k$. Thus the *cost* may be different for different P_r. That is, even though the structure of the final upper triangular matrix $\hat{R}$ does not depend on the choice of the row permutation P_r, the *overall cost* of reducing $\hat{A}$ to upper trapezoidal form *may depend* very much on P_r. A small example is given in Figures 2.3.1 and 2.3.2 to illustrate the effect of the choice of P_r.

Hence in general we want to choose a row permutation P_r so that the cost of reducing P_rAP_c is small. The following algorithm is the same as Algorithm 2.2.1 with the addition of choosing a row permutation.

$$\overline{A} = \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ \times & 0 & 0 & \times & 0 \\ \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & 0 & \times \\ 0 & 0 & \times & \times & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \end{pmatrix}$$

$$\overline{R}^1 = \underset{cost=2}{\begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}} \qquad \overline{R}^2 = \underset{cost=5}{\begin{pmatrix} * & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}}$$

$$\overline{R}^3 = \underset{cost=6}{\begin{pmatrix} * & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}} \qquad \overline{R}^4 = \underset{cost=2}{\begin{pmatrix} \times & \times & 0 & \times & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & * & 0 & * \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}}$$

$$\overline{R}^5 = \underset{cost=6}{\begin{pmatrix} \times & \times & 0 & \times & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}} \qquad \overline{R}^6 = \underset{cost=5}{\begin{pmatrix} \times & \times & 0 & \times & 0 \\ 0 & * & 0 & * & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}}$$

$$\overline{R}^7 = \underset{cost=3}{\begin{pmatrix} \times & \times & 0 & \times & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}}$$

$$total \quad cost = 29$$

Figure 2.3.1 The cost of reducing a matrix to upper trapezoidal form.
Nonzeros involved in the reduction are denoted by *.
Nonzeros not involved in the reduction are denoted by $\times$.

$$\hat{A} = \begin{pmatrix} 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 \\ 0 & 0 & \times & 0 & \times \\ \times & \times & 0 & 0 & 0 \\ \times & 0 & 0 & \times & 0 \end{pmatrix}$$

$$\underset{cost=1}{\hat{R}^1 =} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \underset{cost=1}{\hat{R}^2 =} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\underset{cost=2}{\hat{R}^3 =} \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \underset{cost=2}{\hat{R}^4 =} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\underset{cost=6}{\hat{R}^5 =} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix} \qquad \underset{cost=3}{\hat{R}^6 =} \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

$$\underset{cost=8}{\hat{R}^7 =} \begin{pmatrix} * & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}$$

$$total \quad cost = 23$$

Figure 2.3.2 The cost of reducing a matrix to upper trapezoidal form. (The matrix is a permuted form of that in Figure 2.3.1.)

Algorithm 2.3.1 (orthogonal reduction of sparse matrix - with row and column permutations)

(1) Determine the nonzero structure of A^TA.

(2) Choose a column permutation for A so that the Cholesky factor $\overline{R}$ of $P_c^T(A^TA)P_c$ is sparse.

(3) Perform a symbolic factorization on $P_c^T(A^TA)P_c$ to determine the nonzero structure of $\overline{R}$ and set up an appropriate storage scheme for $\overline{R}$.

(4) Determine a good row permutation P_r for AP_c.

(5) Use Algorithm 2.1.1 to rotate the rows of P_rAP_c sequentially into $\overline{R}$.

There appears to be no literature on the problem of finding good row permutations in this context. In the next few chapters we will introduce a graph model for studying this problem and describe two heuristic algorithms for finding good row permutations for row elimination.

2.4. A storage scheme for sparse matrices

For completeness we describe a storage scheme for the n by n sparse upper triangular matrix R. Other storage schemes for sparse matrices can be found in [36, 49, 63].

Consider the elimination of a^k using Algorithm 2.1.1. Exactly one row of R^k (or R) is needed in the annihilation of a nonzero in a^k. Thus one must be able to access the nonzeros of *any row* of R from the data structure easily and efficiently. A simple storage scheme is the following.

(1) The diagonal elements of R are stored in a floating-point array DIAG.

(2) The off-diagonal nonzeros in each row of R are stored consecutively in a floating-point array RNZ. An integer array NZSUB is used to record the column index of each nonzero in RNZ. An additional integer array XRNZ is used to store pointers to the beginning of the off-diagonal nonzeros in each row of R.

DIAG	-	R_{11}	R_{22}	R_{33}	R_{44}	R_{55}	
RNZ	-	R_{12}	R_{14}	R_{24}	R_{34}	R_{35}	R_{45}
XRNZ	-	1	3	4	6	7	7
NZSUB	-	2	4	4	4	5	5

Figure 2.4.1 Storage scheme for the last upper triangular matrix shown in Figures 2.3.1 and 2.3.2.

An example is given in Figure 2.4.1. The upper triangular matrix is the last upper triangular matrix shown in Figures 2.3.1 and 2.3.2.

Assume that an integer and a floating-point number occupy the same amount of space, say one storage location. Then the number of storage locations required in this storage scheme is

(1) DIAG - n locations,

(2) RNZ - $|R|$ locations,

(3) NZSUB - $|R|$ locations, and

(4) XRNZ - $n+1$ locations (for programming convenience, XRNZ($n+1$) has the value $|R|+1$).

The total is $2|R|+2n+1$.

In the scheme described above, the length of NZSUB is the same as that of RNZ. However NZSUB may contain redundant information (see the example in Figure 2.4.1). It is

DIAG	-	R_{11}	R_{22}	R_{33}	R_{44}	R_{55}	
RNZ	-	R_{12}	R_{14}	R_{24}	R_{34}	R_{35}	R_{45}
XRNZ	-	1	3	4	6	7	7
NZSUB	-	2	4	5			
XNZSUB	-	1	2	2	3	3	

Figure 2.4.2 Compressed storage scheme for the example shown in Figure 2.4.1.

often true that the column indices of the nonzeros in a row of R form a final subsequence of those of a previous row. Thus NZSUB can be "compressed" by removing this redundant information. On the other hand, we now need an extra integer array XNZSUB to store pointers to the beginning of column indices in NZSUB for each row. This compressed storage scheme is due to Sherman [68]. The compressed storage scheme for the example in Figure 2.4.1 is given in Figure 2.4.2. Experience has shown that the storage required by this compressed storage scheme is usually much smaller than that required by the uncompressed scheme, especially for very large problems.

CHAPTER 3

GRAPH-THEORETIC RESULTS

Many sparse matrix algorithms are formulated as graph algorithms, since sparse matrix problems can be described conveniently in terms of graphs. This is especially true in finding a symmetric row and column ordering for a sparse symmetric positive definite matrix, which is formulated as a problem of relabelling the nodes in an undirected graph. Our problem is to find good column and row permutations for an m by n matrix A, with $m \geq n$. As we have noted in the previous chapter, finding a good column ordering for A is the same as finding a good symmetric row and column ordering for the symmetric positive definite matrix $A^T A$. Thus our problem is partially solved, since there already exist good heuristic algorithms for finding good symmetric orderings for symmetric positive definite matrices. What we have to do is to develop algorithms for finding good row orderings for A. In this chapter we introduce a graph model for studying row elimination using rotations. This model is similar to the one used for sparse Cholesky decomposition. Some graph-theoretic results relating row and column orderings are then obtained. The matrix version of some of the results appears in [38].

3.1. Basic graph-theoretic terminology

An *undirected graph* $G=(X,E)$ consists of a finite set X of *nodes* (or *vertices*) together with a set E of *edges* which are unordered pairs of distinct nodes. A graph $\overline{G}=(\overline{X},\overline{E})$ is a *subgraph* of G if $\overline{X} \subseteq X$ and $\overline{E} \subseteq E \bigcap (\overline{X} \times \overline{X})$. For any non-empty subset Y of X, the *section subgraph* $G(Y)$ *induced by* Y is the subgraph $(Y,E(Y))$ of G, where $E(Y)=\{\, \{x,y\} \in E \mid x,y \in Y \,\}$.

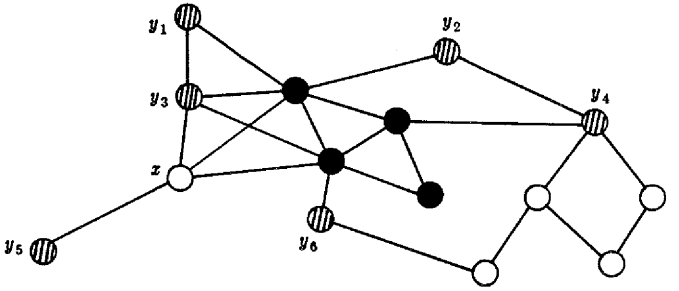
A node y is said to be *adjacent* to another node x in G if $\{x,y\} \in E$. The node y is sometimes called a *neighbor* of x. The *degree* of a node x is the number of its neighbors. For $Y \subseteq X$, the *adjacent set* of Y is defined as

$$Adj(Y)=\{\, x\in X-Y \mid \{x,y\}\in E \text{ for some } y\in Y \,\} \ .$$

If $Y=\{y\}$, we write $Adj(y)$ rather than $Adj(\{y\})$.

A set $C\subseteq X$ is a *clique* of G if the nodes in C are pairwise adjacent; that is, if $x,y\in C$, then $\{x,y\}\in E$. The section subgraph $G(C)$ is called a *complete subgraph.*

For distinct nodes x and y in G, a *path* from x to y of length l is an ordered set of distinct nodes $(v_0,v_1,\cdots,v_l)$, where $v_0=x$, $v_l=y$, and $v_i\in Adj(v_{i-1})$, $1\leq i\leq l$. The graph G is said to be *connected* if there is at least one path connecting every pair of distinct nodes in G. If G is *disconnected*, it consists of *two or more* connected subgraphs called *components.* The *distance* $d(x,y)$ between any two nodes x and y in a connected graph G is the length of the shortest path connecting them.



Nodes that are darkened are in V.
Nodes that are shaded are in $Reach_G(x,V)$.

Figure 3.1.1 An example of reachable set.

Let $V \subseteq X$ and $y \notin V$. The node y is said to be *reachable from* a node x *through* V if there exists a path $(x, v_1, \cdots, v_k, y)$ from x to y such that $v_i \in V$ for $1 \leq i \leq k$. The *reachable set* of x *through* V is then defined to be the set

$$Reach_G(x,V) = \{\, y \in X-V \mid y \text{ is reachable from } x \text{ through } V \,\} \ .$$

Note that the paths may be only of length one (k may be zero). That is, any node $y \notin V$ that is in $Adj(x)$ is also reachable from x through V. Furthermore, V may be empty, in which case $Reach_G(x,V) = Adj(x)$. An example is shown in Figure 3.1.1. Nodes that are in V are darkened and nodes that are in $Reach_G(x,V)$ are shaded.

Let $G=(X,E)$ be a connected graph. A non-empty subset S of X is a *separator* of G if the section subgraph $G(X-S)$ consists of *two or more* components; that is, $G(X-S)$ is disconnected. We denote the components by $G(C_1)$, $G(C_2)$, $\cdots$, $G(C_k)$, $k \geq 2$. The set S is called a *width-l separator* of G (or an *l-wide separator* [40]) if for $x \in C_i$ and $y \in C_j$, $i \neq j$, the distance $d(x,y)$ in G is *greater than* l. Furthermore, if no proper subset of S is a width-l

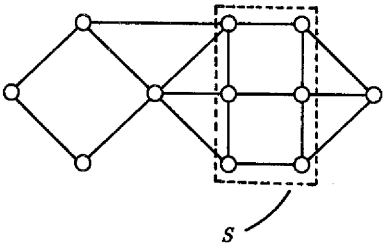


Figure 3.1.2 An example of a minimal width-2 separator (S).

separator of G, then S is called a *minimal width-l separator* of G. In this thesis, we are concerned with the cases $l=1$ and $l=2$. An example of a width-l separator is given in Figure 3.1.2, for $l=2$. Liu considered the use of width-2 separators when he investigated the solution of sparse systems on parallel machines [58]. Gilbert also used width-2 separators in the solution of sparse systems using Gaussian elimination with partial pivoting [40].

Let $G_1=(X_1,E_1)$ and $G_2=(X_2,E_2)$ be two graphs. The *union* of G_1 and G_2, denoted by $G_1 \bigcup G_2$, is the graph $(X_1 \bigcup X_2, E_1 \bigcup E_2)$. An example is given in Figure 3.1.3.

For a graph $G=(X,E)$ with $|X|=n$, an *ordering* (or *labelling*) of G is a bijective mapping

$$\alpha : \{1,2,\cdots,n\} \rightarrow X \ .$$

We denote a graph G with an ordering α by G^α. The node and edge sets of G^α are denoted by

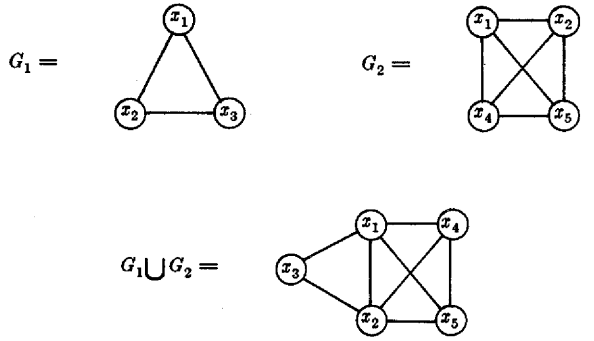


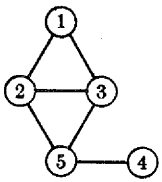
Figure 3.1.3 An example of the union of two graphs.

X^{α} and E^{α} respectively. Suppose α_1 and α_2 are two different labellings of G. It should be noted that the graphs G^{α_1} and G^{α_2} have the same *structure*. In fact, the set of labelled graphs $\{G^{\alpha}\}$ forms an equivalence class.

3.2. Computer representation of graphs

As we see in the next section and Chapters 4 and 5, the column and row ordering problems are expressed respectively as labelling the nodes of an undirected graph and merging of graphs. Thus it is important that graphs can be represented economically and efficiently so that these operations on graphs can be performed easily.

Let $G=(X,E)$ be an undirected graph with n nodes. In most cases, we have to access the adjacent sets frequently. Therefore it is appropriate to store them consecutively in an integer array, say ADJNCY. Then an integer array, XADJ, is used to store pointers to the beginning of each adjacent set. An example is given in Figure 3.2.1. If G has r edges, then ADJNCY will



XADJ	-	1	3	6	9	10	13						
ADJNCY	-	2	3	1	3	5	1	2	5	5	2	3	4

Figure 3.2.1 A representation of a graph.

have $2r$ entries. For programming convenience, XADJ($n+1$) contains $2r+1$. Hence G is represented using $n+2r+1$ integers. Note that the neighbors of node k are contained in ADJNCY(i), for $i =$ XADJ(k), XADJ(k)+1, $\cdots$, XADJ($k+1$)–1. This representation is the same as that used in SPARSPAK [30]. Other representations of graphs are described in [36, 48].

3.3. Graph model for row elimination and basic results

Let A be an m by n matrix with $m \geq n$, and R^k be the n by n upper triangular matrix obtained by eliminating the first k rows of A using rotations. We assume A^TA is sparse. Denote the k–th row of A by a^k. In Chapter 2 we have shown that the cost of eliminating a^k depends on the structures of a^k and R^{k-1}. Implicitly, the cost depends on the structures of the first k rows of A (that is, the row ordering). Thus a model for studying the row elimination (and the row ordering) problem should possess the following characteristics.

(1) It models the structures of the sequence of upper triangular matrices R^0, R^1, $\cdots$, R^{m-1}, R^m.

(2) It provides a way to simulate the process of rotating a row, say a^k, into R^{k-1}.

Note that the sequence of upper triangular matrices "approaches" R^m, which is the Cholesky factor of the sparse symmetric positive definite matrix A^TA. Thus a reasonable model to be used for studying the row elimination problem would be one that is based on a model for sparse Cholesky decomposition. Because of this observation we will first present a model for sparse Cholesky factorization. In the following discussion, if M is a matrix, M_{ij} denotes the (i,j)-element of M, and $|M|$ denotes the number of nonzeros in M. Also $|S|$ denotes the number of elements in S if S is a set.

Let B be an n by n sparse symmetric positive definite matrix. The *graph* of B, denoted by $G_B=(X_B,E_B)$, is a labelled undirected graph with $X_B=\{x_1,x_2,\cdots,x_n\}$ and $\{x_i,x_j\}\in E_B$ if and only if $B_{ij}\neq 0$, $i\neq j$. Here x_i is the node having labelling i. An example is given in Figure 3.3.1. Sparse Cholesky decomposition can be described using the graph of B. See [36, 61, 64, 72]

for more details.

Let R be the Cholesky factor of B. Lemma 3.3.1 tells us exactly where fill-in will occur in sparse Cholesky decomposition from the graph of B [33, 65]. Note that it does not depend on the numerical values of the nonzero elements in B. Thus it allows one to predict, before any numerical computation begins, the nonzero structure of R. This forms the basis for symbolic factorization [31, 35].

Lemma 3.3.1

Let $S_i=\{x_1,x_2,\cdots,x_{i-1}\}$. Then for $j>i$, $R_{ij}\neq 0$ if and only if $x_j\in Reach_{G_B}(x_i,S_i)$.

□

Consequently the number of off-diagonal nonzeros in R is $\rho=\sum_{i=1}^{n}\left|Reach_{G_B}(x_i,S_i)\right|$.

Suppose P is any n by n permutation matrix. Let $\overline{B}=P^TBP$, and $G_{\overline{B}}=(X_{\overline{B}},E_{\overline{B}})$ be the graph of $\overline{B}$. Let $X_{\overline{B}}=\{\overline{x}_1,\overline{x}_2,\cdots,\overline{x}_n\}$, where $\overline{x}_i$ is the node having labelling i in $G_{\overline{B}}$. The

$$B=\begin{pmatrix} \times & 0 & 0 & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 & \times \\ 0 & \times & \times & 0 & \times & 0 & 0 \\ \times & 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & 0 \\ 0 & \times & 0 & 0 & \times & 0 & \times \end{pmatrix}$$

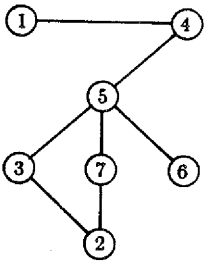


Figure 3.3.1 Graph of a sparse symmetric positive definite matrix.

structures of G_B and $G_{\overline{B}}$ are the *same*, but their *labellings* are *different*. An example is given in Figure 3.3.2 in which the matrix B is the same as that in Figure 3.3.1.

Let $\overline{R}$ be the Cholesky factor of $\overline{B}$. Then the number of off-diagonal nonzeros in $\overline{R}$ is $\overline{\rho}=\sum_{i=1}^{n} \left| Reach_{G_{\overline{B}}}(\overline{x}_i,\overline{S}_i) \right|$, where $\overline{S}_i=\{\overline{x}_1,\overline{x}_2,\cdots,\overline{x}_{i-1}\}$. Clearly ρ and $\overline{\rho}$ may be different since $\left| Reach_{G_B}(x_i,S_i) \right|$ and $\left| Reach_{G_{\overline{B}}}(\overline{x}_i,\overline{S}_i) \right|$ may not be the same. In other words $|R| \neq |\overline{R}|$ in general (as we have seen in Chapter 2). Hence the problem of finding a "good" permutation for a sparse symmetric positive definite matrix B can be stated as follows. Given the graph G_B of B, relabel the nodes of G_B such that the reachable sets $Reach_{G_{\overline{B}}}(\overline{x}_i,\overline{S}_i)$ are small. A thorough

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\overline{B} = \begin{pmatrix} \times & 0 & \times & 0 & 0 & \times & 0 \\ 0 & \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 & \times \\ 0 & 0 & \times & 0 & \times & \times & 0 \\ \times & 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & \times & 0 & 0 & \times \end{pmatrix}$$

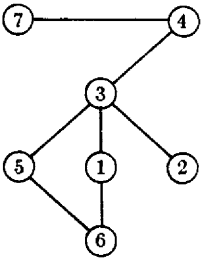


Figure 3.3.2 Graph of $\overline{B}=P^TBP$, where
B is the same as the one given in Figure 3.3.1.

treatment of this problem can be found in [36].

Now we return to our row elimination problem. Denote the j–th element of row a^k by a_j^k. For each row a^k of A, define an n by n symmetric matrix $Y^k=(a^k)^T(a^k)$. Note that $Y_{ij}^k\neq 0$ if and only if a_i^k and a_j^k are *both* nonzero. Moreover some of the columns and rows of Y^k may be null; they correspond to the zeros in a^k. In fact if *all* the null columns and null rows are deleted from Y^k, what is left will be a dense square matrix whose order is the same as the number of nonzeros in a^k.

The *row graph* of a^k, denoted by $\phi^k=(\chi^k,\epsilon^k)$, is a labelled undirected graph with $\chi^k=\{x_i \mid a_i^k\neq 0\}$ and $\epsilon^k=\{\{x_i,x_j\} \mid x_i,x_j\in\chi^k\}$. Hence the row graph of any row of A is a complete graph and it corresponds to the dense submatrix in Y^k. The row graph ϕ^k of a^k will sometimes be called the graph of Y^k. An example is given in Figure 3.3.3.

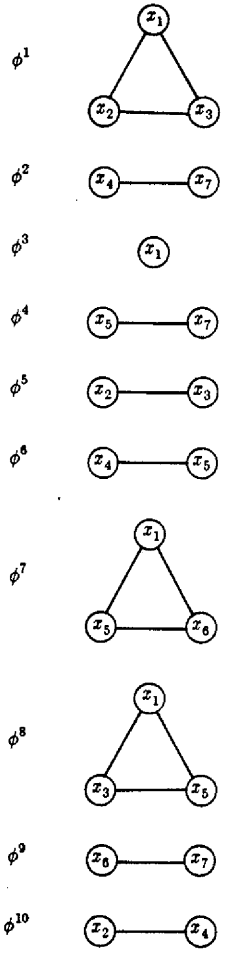
Recall that the cost of eliminating a^k depends on the structures of the first k rows of A, so it is helpful to consider the k by n matrix A^k which is defined by

$$A^k = \begin{pmatrix} a^1 \\ a^2 \\ \cdot \\ \cdot \\ \cdot \\ a^k \end{pmatrix} .$$

The upper triangular matrix R^k is obtained by eliminating the rows of A^k using rotations. Let B^k denote the n by n symmetric matrix $(A^k)^T(A^k)$. Note that

$$B^k = \begin{pmatrix} (a^1)^T & (a^2)^T & \cdots & (a^k)^T \end{pmatrix} \begin{pmatrix} a^1 \\ a^2 \\ \cdot \\ \cdot \\ \cdot \\ a^k \end{pmatrix} = \sum_{l=1}^{k} (a^l)^T(a^l) = \sum_{l=1}^{k} Y^l \ .$$

That is, the symmetric matrix B^k is the *sum* of k sparse matrices, each of which contains a dense submatrix. Lemma 3.3.2 follows directly from the definition of B^k.



$$A = \begin{bmatrix} \times & \times & \times & & & & \\ \times & & & \times & & & \times \\ & & & & \times & & \times \\ & \times & \times & & & & \\ & & & \times & \times & & \\ \times & & & & \times & \times & \\ \times & & \times & & \times & & \\ & & & & & \times & \times \\ & \times & & \times & & & \end{bmatrix}$$

Figure 3.3.3 The sequence of row graphs of an 10 by 7 matrix.

Lemma 3.3.2

Assume exact cancellation does not occur. Then $B_{ij}^k \neq 0$ if and only if $a_i^l \neq 0$ and $a_j^l \neq 0$ for some $l \leq k$.

□

Note that there may be some null columns in the matrix A^k. Thus as in the case of Y^l, B^k may be *structurally singular*, since any null columns in A^k correspond to null rows and null columns in B^k.

Define a labelled undirected graph $G^k=(X^k,E^k)$ as follows. Let

$$X^k = \{x_i \mid \text{column } i \text{ and row } i \text{ of } B^k \text{ are non-null}\} \ ,$$

and

$$E^k = \{\{x_i,x_j\} \mid B_{ij}^k \neq 0\} \ .$$

The sequence of G^k corresponding to the example given in Figure 3.3.3 is shown in Figure 3.3.4. Let W^k be the submatrix obtained from B^k by deleting *all* the null rows and null columns. Thus G^k is the graph of W^k. Assume that the non-null columns of A are linearly independent. Then W^k is *symmetric and positive definite.*

Furthermore G^k can be defined using the row graphs of a^1, a^2, $\cdots$, and a^k.

Lemma 3.3.3

$G^k = \bigcup_{l=1}^{k} \phi^l$. That is, $X^k = \bigcup_{l=1}^{k} \chi^l$ and $E^k = \bigcup_{l=1}^{k} \epsilon^l$.

Proof:

Suppose $x_i,x_j \in X^k$ and $\{x_i,x_j\} \in E^k$. Then by the definition of G^k, $B_{ij}^k \neq 0$. Applying Lemma 3.3.2, there must exist some $l \leq k$ such that $a_i^l \neq 0$ and $a_j^l \neq 0$. Hence by the definition of row graphs, $x_i,x_j \in \chi^l$ and $\{x_i,x_j\} \in \epsilon^l$.

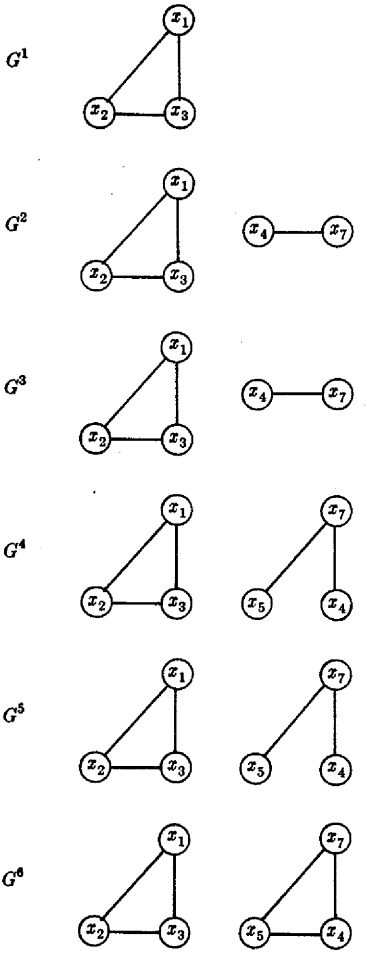


Figure 3.3.4(a) The sequence of G^k for the matrix shown in Figure 3.3.3.

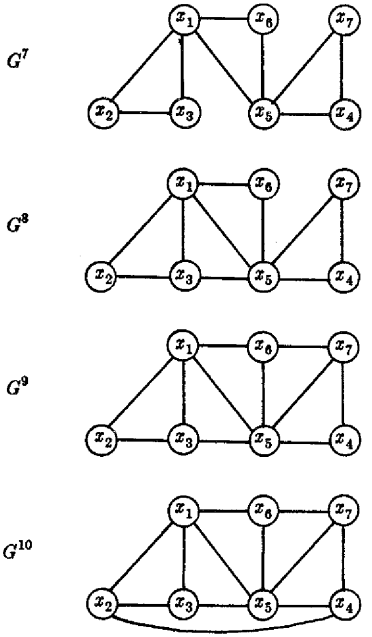


Figure 3.3.4(b) Continuation of Figure 3.3.4(a).

On the other hand, if $x_i,x_j \in X^l$ and $\{x_i,x_j\} \in \epsilon^l$, for $l \leq k$, then a_i^l and a_j^l are both nonzero. Lemma 3.3.2 and the definition of G^k immediately imply that $x_i,x_j \in X^k$ and $\{x_i,x_j\} \in E^k$.

□

Note that when $k=m$ we have $B^m=(A^m)^T(A^m)=A^TA$ and $G^m=\bigcup_{l=1}^{m} \phi^l$ is the graph of A^TA. Thus, assuming exact cancellation does not occur, the nonzero structure of the Cholesky factor R^m (or R) of the symmetric positive definite matrix A^TA can be obtained from G^m using Lemma 3.3.1. How about the structure of R^k when $k<m$? It is known that B^k may be

structurally singular since it may have null columns and null rows. But any null columns and null rows in B^k must correspond to null columns and null rows in R^k. In fact if V^k is the submatrix obtained from R^k by deleting the null columns and null rows, then V^k is structurally the Cholesky factor of W^k. Thus, apart from the null columns and null rows in R^k, the nonzero structure of R^k can be determined from G^k using Lemma 3.3.1. Here instead of using S_i, we have S_i^k which includes only those nodes that are actually in G^k. This is summarized in the following lemma which is a generalization of Lemma 3.3.1.

Lemma 3.3.4

For $j>i$, $R_{ij}^k \neq 0$ if and only if $x_i, x_j \in X^k$ and $x_j \in Reach_{G^k}(x_i, S_i^k)$ where $S_i^k = \{x_l \in X^k \mid l<i\}$.

□

Thus the structures of the upper triangular matrices $R^1, R^2, \cdots, R^m$ can be modelled by the sequence of graphs $G^1, G^2, \cdots, G^m$. It is important to note that our discussion and Lemmas 3.3.1 and 3.3.4 assume that we are working with B^k and that *exact cancellation does not occur* during the computation. Furthermore *all* elimination sequences are assumed to be *maximal*. The structure of R^k, $k=1,2,\cdots,m$, determined only represents the *worst case* situation. This is illustrated by two examples in Figures 3.3.5 and 3.3.6. If $m \gg n$, there would be more maximal elimination sequences than non-maximal ones. Thus in this case the actual structure of R should be very close to that predicted by Lemma 3.3.1. We will see later in this thesis that this is indeed the case.

We now consider the graph G^k in more detail. By Lemma 3.3.3, we have

$$G^k = \bigcup_{l=1}^{k} \phi^l = \left\{ \bigcup_{l=1}^{k-1} \phi^l \right\} \bigcup \phi^k = G^{k-1} \bigcup \phi^k \ .$$

Thus from the graph-theory point of view, eliminating a row is equivalent to *merging the row graph ϕ^k with the existing graph G^{k-1} of B^{k-1}*. Lemma 3.3.5 is equivalent to Lemma 2.1.4. It

$$A^3 = \begin{pmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

$$B^3 = \begin{pmatrix} \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

$$\text{Expected structure of } R^3 = \begin{pmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

$$\text{Actual structure of } R^3 = \begin{pmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

Figure 3.3.5 Difference between the expected and the actual structures of R^k, for k=3.

relates the elimination sequence we introduced in Chapter 2 to our graph model. It also shows that the elimination sequence depends on the structure of G^k. Note that we are assuming that exact cancellation does not occur and all elimination sequences are maximal.

Lemma 3.3.5

Let $\Xi^k = \{\xi_1^k, \xi_2^k, \cdots, \xi_{\mu_k}^k\}$ be the elimination sequence of row a^k. Then

(1) $\xi_1^k = \min\{l \mid x_l \in X^k\}$.

(2) For $1 \le i < \mu_k$, $\xi_{i+1}^k = \min\{l \mid x_l \in Reach_{G^k}(x_{\xi_i^k}, S_{\xi_i^k}^k)\}$ where $S_{\xi_i^k}^k = \{x_j \in X^k \mid j < \xi_i^k\}$.

Proof:

(1) Since ϕ^k is the row graph of a^k, the first nonzero in a^k corresponds to the node x_p such that $p = \min\{l \mid x_l \in X^k\}$.

(2) By Lemma 3.3.4, the off-diagonal nonzeros in row ξ_i^k of R^k are given by the set $Reach_{G^k}(x_{\xi_i^k}, S_{\xi_i^k}^k)$. Hence the result follows.

□

The next lemma, which is equivalent to Lemma 2.1.6, characterizes maximal elimination sequences using the graph model.

Lemma 3.3.6

Let Ξ^k be a maximal elimination sequence and $s \in \Xi^k$. If $x_t \in Reach_{G^k}(x_s, S_s^k)$, then $t \in \Xi^k$.

□

In Chapter 2 the cost of eliminating a^k is defined to be $\sum_{p \in \Xi^k} \lambda^k(p)$, where $\lambda^k(p)$ is the number of nonzeros in row p of R^k. This can be restated using our graph model: the cost of merging a row graph ϕ^k with G^{k-1} is

$$\sum_{p \in \Xi^k} \left\{ \left| Reach_{G^k}(x_p, S_p^k) \right| + 1 \right\} .$$

Note that this cost depends on the structure of G^k which depends, in turn, on the structures of the row graphs $\phi^1, \phi^2, \cdots, \phi^k$. Thus the problem of finding a "good" row ordering for A can now be viewed as the problem of arranging the row graphs such that the cost of merging the row graphs is small.

Some basic results about the row elimination (or graph merging) process can be derived easily from this graph model.

$$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{pmatrix} \qquad B = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}$$

$$\text{Expected structure of } R = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{pmatrix}$$

$$\text{Actual structure of } R = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{pmatrix}$$

Figure 3.3.6 Another example illustrating the difference between the expected and the actual structures of R.

Let $\psi^k=\{C_1^k, C_2^k, \cdots, C_{\gamma_k}^k\}$ be the *component partitioning* of G^k. That is, the partitioning of the node set X^k induced by the connected components $G^k(C_j^k)$ of the graph G^k. Thus for $1\leq i,j\leq\gamma_k$ and $i\neq j$, $C_i^k \bigcap C_j^k=\emptyset$ and $\bigcup_{l=1}^{\gamma_k} C_l^k=X^k$. Furthermore no path joins any node in C_i^k and any node in C_j^k when $i\neq j$. (See Figure 3.3.4 for an example.) Lemma 3.3.7 is a consequence of Lemma 3.3.3.

Lemma 3.3.7

For $1\leq k_1<k_2\leq m$, if $C_i^{k_1}\in\psi^{k_1}$, then there exists j such that $C_i^{k_1}\subseteq C_j^{k_2}$ and $C_j^{k_2}\in\psi^{k_2}$.

□

That is, the size of any component set, say C_i^k, is non-decreasing as k increases.

It is important to note that the sequence of component partitionings $\{\psi^k\}$ depends *only* on the order in which the row graphs are merged; that is, it depends on the row ordering of A. The effect of permuting the columns of A is just a relabelling of the nodes of G^k which may, however, affect the cost of eliminating the rows, and the nonzero structures of the upper triangular matrices.

The next result illustrates the significance of the component partitioning ψ^k.

Theorem 3.3.8

Let $\psi^k=\{C_1^k, C_2^k, \cdots, C_{\eta_k}^k\}$ be the component partitioning of G^k. Denote the component containing the row graph ϕ^k by $G^k(C_{\sigma_k}^k)$. Then $x_p \in C_{\sigma_k}^k$, for all $p \in \exists^k$.

Proof:

The proof is by contradiction. Since $x_{\xi_1^k} \in C_{\sigma_k}^k$, there must exist two consecutive members s and t of $\exists^k$ such that $x_s \in C_{\sigma_k}^k$ and $x_t \in C_l^k$, for some $l \neq \sigma_k$. From Lemma 3.3.5, $t=\min\{i \mid x_i \in Reach_{G^k}(x_s, S_s^k)\}$. This means that there must be a path joining x_s and x_t in G^k. This contradicts the fact that x_s and x_t are in different components.

□

Theorem 3.3.8 shows that the set of nodes that are involved in the elimination of row k,

$$\{x_{\xi_1^k}, x_{\xi_2^k}, \cdots, x_{\xi_{\mu_k}^k}\} ,$$

is limited to the component $G^k(C_{\sigma_k}^k)$ whose node set $C_{\sigma_k}^k$ contains χ^k, the node set of the row graph ϕ^k. Note that the cost of eliminating a row depends in part on the *length* of the elimination sequence. Hence we want to find a row ordering which allows the component $G^k(C_{\sigma_k}^k)$ to be kept small for as large a k as possible.

Theorem 3.3.9

The cost of eliminating row k is bounded by

$$\frac{1}{2} \mid C_{\sigma_k}^k \mid \left(\mid C_{\sigma_k}^k \mid + 1 \right) .$$

Proof:

The bound is obtained by assuming Ξ^k is maximal and for $i \in \Xi^k$, row i of R^k has nonzeros in position j wherever $x_j \in C_{\sigma_k}^k$ and $j > i$.

□

Corollary 3.3.10

Let Ξ^k be the elimination sequence of row a^k and $\eta = \min\{p \mid p \in \Xi^k\}$. Then $\Xi^k \subseteq \{q \geq \eta \mid x_q \in C_{\sigma_k}^k\}$. Furthermore if $\delta_k = \mid \{q \geq \eta \mid x_q \in C_{\sigma_k}^k\} \mid$, then the cost of eliminating a^k is bounded by $\frac{1}{2}\delta_k(\delta_k + 1)$.

□

Theorem 3.3.9 suggests that we should keep $C_{\sigma_k}^k$ small and Corollary 3.3.10 suggests that regardless of whether $C_{\sigma_k}^k$ is small, we should arrange that the column index of the first nonzero of row k be as large as possible.

3.4. A model problem

In the next two chapters we will describe two algorithms for finding good row and column orderings for a sparse rectangular matrix. The complexity of each algorithm will be analyzed for a model problem. This model problem is defined on an n by n grid. It is typical of those which arise in the natural factor formulation of finite element methods [3, 4, 38].

Consider an n by n grid which consists of $(n-1)^2$ small squares (or elements). An example is given in Figure 3.4.1 with $n=3$. For our purpose, the model problem is defined as follows. Associated with each of the n^2 grid points (or nodes) is a variable, and associated with

each of the $(n-1)^2$ small squares is a set of four equations (or rows) involving the four variables at the corners of the square. This gives rise to a large sparse overdetermined system of linear equations. The number of equations is $4(n-1)^2$ and the number of unknowns is n^2. The unknowns are the variables at the grid points. Denote the coefficient matrix by A. The matrix associated with the example in Figure 3.4.1 is shown in Figure 3.4.2. The matrix A is reduced to upper trapezoidal form using rotations. Let R be the n^2 by n^2 upper triangular matrix obtained after the transformation.

Because of the way in which the model problem is defined, the row graph of any row has exactly four nodes and each node corresponds to a variable at a grid point of a small square. The graph of A^TA, which is the union of all the row graphs, is therefore almost identical to the n by n grid. The only difference is that all the four nodes of a small square are now pairwise connected. Figure 3.4.3 shows the graph of A^TA, where A is the matrix shown in Figure 3.4.2.

In the previous section we have pointed out that the graph model we introduced assumes that all elimination sequences are maximal. However this assumption is not valid in general. For example, the elimination sequence of the first row cannot be maximal unless it contains exactly one nonzero. Thus the graph model only represents the worst case situation. In general the

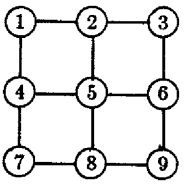


Figure 3.4.1 A 3 by 3 finite element grid.

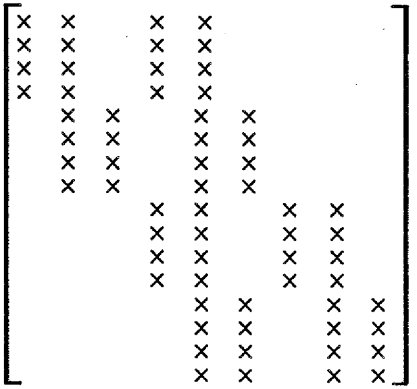


Figure 3.4.2 The matrix associated with a 3 by 3 finite element grid.

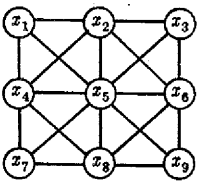


Figure 3.4.3 Graph of a 3 by 3 finite element grid.

number of nonzeros predicted by the graph model using Lemma 3.3.4 (or Lemma 3.3.1) may be larger than the actual number of nonzeros in R. We now show that for our model problem, even

though not all the elimination sequences are maximal, the actual structure of R is given *correctly* by the graph model. That is, the nonzeros are given correctly by Lemma 3.3.1.

Let $G=(X,E)$ be the graph of A^TA. For simplicity we will use the *same notation* for the nodes in G and for the column indices of A. That is, the node set of G is $X=\{1,2,\cdots,n^2\}$.

The following two results will be useful in proving our claim.

Lemma 3.4.1

Let $\Xi^k=\{\xi_1,\xi_2,\cdots,\xi_\mu\}$ be the elimination sequence of a given row a^k. Suppose ξ_s and ξ_t are any two members of Ξ^k with $t>s$. Let G^k be the union of the row graphs of a^1, a^2, $\cdots$, a^k. Then there is a path in G^k joining ξ_s and ξ_t. The nodes on this path are ξ_{s+1}, $\cdots$, ξ_{t-1}, and nodes with labels less than ξ_{t-1}.

Proof:

Let ξ_l and ξ_{l+1} be two consecutive members of Ξ^k. It follows from Lemma 3.3.5 and the definition of reachable sets that there is a path $(\xi_l,\theta_1,\theta_2,\cdots,\theta_p,\xi_{l+1})$ in G^k such that $\theta_i<\xi_l$, for $1\le i\le p$. Now suppose ξ_s and ξ_t are any two members of Ξ^k with $t>s$. Using the observation above repeatedly shows that there is a path connecting ξ_s and ξ_t in G^k and the nodes on this path are ξ_s, ξ_{s+1}, $\cdots$, ξ_{t-1}, ξ_t, and nodes with labels less than ξ_{t-1}.

□

Lemma 3.4.2

Let Ξ^k be a maximal elimination sequence. Suppose s and t are two nodes in the same component in the graph G^k with $s\in\Xi^k$, $t\notin\Xi^k$ and $s<t$. Then any path joining s and t in G^k must contain a node v such that $v>t$.

Proof:

The proof is by contradiction. Assume there is path $(s,v_1,v_2,\cdots,v_r,t)$ such that $v_i<t$, for $1\le i\le r$. Note that $r\ge1$. Otherwise $\{s,t\}$ would be an edge in G^k. This would mean that

$t \in \Xi^k$ because of Lemma 3.3.6. Let v_p be the first node on the path such that $v_p > s$. This implies that $v_p \in Reach_{G^k}(s, S_s^k)$. Because of Lemma 3.3.6, v_p is in the elimination sequence Ξ^k. Using the same argument repeatedly shows that t must be in Ξ^k too, which is a contradiction.

□

We now show that the structure of the upper triangular matrix R is predicted correctly by the graph model. Without loss of generality, we assume that the rows associated with a small square are eliminated *together*. (Note that using a particular row ordering in the proof does not affect the result, since the structure of R depends only on the column permutation of A or the labelling of the nodes in the graph of $A^T A$.)

Suppose the small squares in the n by n grid are numbered from 1 to $(n-1)^2$ in some manner. Let Z_p denote the set of rows associated with the $p-th$ small square. There will be four rows in Z_p, $1 \leq p \leq (n-1)^2$. The rows in Z_p have the same structure and their row graphs are identical. For simplicity, let $\phi^p = (\chi^p, \epsilon^p)$ denote the row graph of each of these four rows. Let $\chi^p = \{\omega_1^p, \omega_2^p, \omega_3^p, \omega_4^p\}$, where $\omega_1^p < \omega_2^p < \omega_3^p < \omega_4^p$ are the column indices of the nonzeros of a row in Z_p.

Let R^p be the upper triangular matrix obtained by eliminating *all* the rows in Z_1, Z_2, $\cdots$, Z_p. Let $G^p = (X^p, E^p) = \bigcup_{l=1}^{p} \phi^l$. We first prove that Lemma 3.3.4 predicts correctly the structure of R^p. We assume exact cancellation does not occur.

Theorem 3.4.3

(1) $R_{ii}^p \neq 0$ if and only if $i \in X^p$.

(2) If $R_{ii}^p \neq 0$, then the off-diagonal nonzeros in row i of R^p are exactly given by $Reach_{G^p}(i, S_i^p)$, where $S_i^p = \{l \in X^p \mid l < i\}$.

Proof:

The proof is by induction on p.

Consider the elimination of the rows of Z_1. There are four rows which have the same nonzero structure. Assume exact cancellation does not occur. Then after these four rows of Z_1 are eliminated, R^1 will contain a full upper triangular submatrix in rows and columns ω_k^1, $1 \leq k \leq 4$. All other elements in R^1 are zero. This full submatrix corresponds to the complete graph $G^1 = \phi^1$. So the theorem is true for $p = 1$.

Assume the theorem is true for p. Now consider the elimination of the rows in Z_{p+1}. There are five different cases.

(a) $X^p \bigcap \chi^{p+1} = \emptyset$.

(b) There is exactly one node in $X^p \bigcap \chi^{p+1}$.

(c) There are exactly two nodes in $X^p \bigcap \chi^{p+1}$.

(d) There are exactly three nodes in $X^p \bigcap \chi^{p+1}$.

(e) $\chi^{p+1} \subset X^p$.

In case (a), since $X^p \bigcap \chi^{p+1} = \emptyset$, rows and columns ω_k^{p+1}, $1 \leq k \leq 4$, of R^p must be null because of the induction hypothesis. Thus eliminating the four rows of Z_{p+1} using orthogonal reductions will not destroy the existing structure of R^p. The elimination process only introduces a new full upper triangular submatrix into R^p. The nonzero structure of R^{p+1} is given by the following. For $i \notin \chi^{p+1}$, $R_{ij}^{p+1} = R_{ij}^p$. For $i \in \chi^{p+1}$, $R_{ij}^{p+1} \neq 0$ if and only if $j \in \chi^{p+1}$ and $j \geq i$. This is illustrated by an example in Figure 3.4.4. Note that G^p and ϕ^{p+1} are disconnected in the graph $G^{p+1} = G^p \bigcup \phi^{p+1}$ and χ^{p+1} is a clique. Thus row i of R^{p+1} is non-null if and only if $i \in X^{p+1} = X^p \bigcup \chi^{p+1}$. For $i \in X^{p+1}$, the off-diagonal nonzeros in row i of R^{p+1} are given by $Reach_{G^{p+1}}(i, S_i^{p+1})$, since

$$Reach_{G^{p+1}}(i, S_i^{p+1}) = Reach_{G^p}(i, S_i^p) \quad , \quad \text{for } i \notin X^p$$

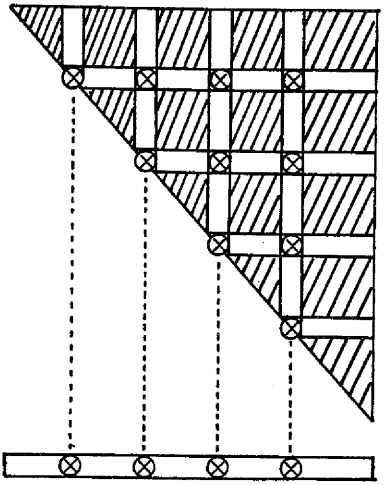


Figure 3.4.4 Elimination of the rows of Z_{p+1}, where $X^p \bigcap \chi^{p+1} = \emptyset$.

and

$$Reach_{G^{p+1}}(i, S_i^{p+1}) = \{\omega_j^{p+1} \in \chi^{p+1} \mid \omega_j^{p+1} > i\} \quad , \quad \text{for } i \in \chi^{p+1} \ .$$

In case (b), let $\{\omega_q^{p+1}\} = X^p \bigcap \chi^{p+1}$. By the induction hypothesis, rows and columns s of R^p must be null, for $s \in \chi^{p+1} - \{\omega_q^{p+1}\}$. Moreover $R_{ij}^p = 0$ for $i \in X^p$, $j \in \chi^{p+1} - \{\omega_q^{p+1}\}$, and $j > i$. This is illustrated by an example in Figure 3.4.5.

Let $\Delta=\{\delta_1,\delta_2,\cdots,\delta_r\}$ where

(i) $\delta_1=\omega_q^{p+1}$, and

(ii) for $1\leq i<r$, δ_{i+1} is the column index of the first off-diagonal nonzero in row δ_i of R^p. In terms of the graph G^p,

$$\delta_{i+1}=\min\{l\,|\,l\in Reach_{G^p}(\delta_i,S_{\delta_i}^p)\}\ .$$

Because of Lemma 2.1.4 and the fact that row and column s of R^p are null for $s\in X^{p+1}-\{\omega_q^{p+1}\}$,

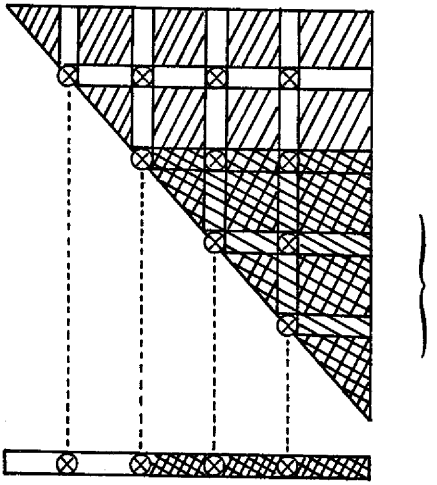


Figure 3.4.5 Elimination of the rows of Z_{p+1}, where $X^p\bigcap X^{p+1}$ contains exactly one node.

the elimination sequence of each row of Z_{p+1} must be a subset of $\Delta \bigcup \chi^{p+1}$. Furthermore since all the rows of Z_{p+1} have the same structure and they are eliminated together, $\Delta \bigcup \chi^{p+1}$ is indeed the elimination sequence of the fourth row and it is maximal.

Let $G^{p+1} = G^p \bigcup \phi^{p+1}$. Consider the structure of R^{p+1}.

(1) $i \notin X^p \bigcup \chi^{p+1}$:

By the induction hypothesis, both row and column i of R^p are null. In the elimination process, row i will not be used as a pivot row. Thus row i will remain null in R^{p+1}. Note that i is not a node in G^{p+1}.

(2) $i < \omega_q^{p+1}$ and $i \in X^p$:

Row i of R^p will not be used as a pivot row in the elimination process. Thus row i will not be affected; that is, $R_{ij}^{p+1} = R_{ij}^p$, for $j \geq i$. By the induction hypothesis, the column indices of the off-diagonal nonzeros in row i are given by $Reach_{G^p}(i, S_i^p)$. But

$$Reach_{G^{p+1}}(i, S_i^{p+1}) = Reach_{G^p}(i, S_i^p) \ ,$$

since $i < \omega_q^{p+1}$ and X^p and χ^{p+1} have exactly one node, ω_q^{p+1}, in common.

(3) $i < \omega_q^{p+1}$ and $i \in \chi^{p+1}$:

Row i of R^p is null since $i \notin X^p$. Thus the elimination process will introduce fill-in in position j, for $j \in \chi^{p+1}$ and $j \geq i$. That is, $R_{ij}^{p+1} \neq 0$ for $i, j \in \chi^{p+1}$ and $j \geq i$. In the graph G^{p+1},

$$Reach_{G^{p+1}}(i, S_i^{p+1}) = \{ l \in \chi^{p+1} \mid l > i \} \ .$$

(4) $i = \omega_q^{p+1}$:

Row i of R^p is not null and it is used as a pivot row. The elimination of the rows of Z_{p+1} can only introduce fill-in in position j, for $j \in \chi^{p+1}$ and $j > \omega_q^{p+1}$. Thus $R_{ij}^{p+1} \neq 0$ if $R_{ij}^p \neq 0$ or $j \in \{ l \in \chi^{p+1} \mid l > \omega_q^{p+1} \}$. Note that in G^{p+1},

$$Reach_{G^{p+1}}(i, S_i^{p+1}) = Reach_{G^p}(i, S_i^p) \bigcup \{ l \in \chi^{p+1} \mid l > \omega_q^{p+1} \} \ ,$$

because ω_q^{p+1} is the only node which is in both X^p and χ^{p+1}.

(5) $i>\omega_q^{p+1}$, $i\in X^p$ and $i\in\Delta$:

By the induction hypothesis, $R_{ij}^p=0$ for $j\in\chi^{p+1}$ and $j>i$. Row i will be used as a pivot row in the elimination process. Because of the structure of the rows of Z_{p+1}, fill-in will only occur in R_{ij}^p for $j\in\chi^{p+1}$ and $j>i$. That is, $R_{ij}^{p+1}\neq 0$ if $R_{ij}^p\neq 0$ or $j\in\{l\in\chi^{p+1}\,|\,l>i\}$. Consider the graph G^{p+1}. By Lemma 3.4.1 there is a path $(\omega_1^{p+1},v_1,v_2,\cdots,v_r,i)$ in G^{p+1} such that $v_i<i$, $1\leq i\leq r$. Thus any node in χ^{p+1} whose label is greater than i is also reachable from i through nodes whose labels are less than i. Since ω_q^{p+1} is the only node that is in both X^p and χ^{p+1}, we have

$$Reach_{G^{p+1}}(i,S_i^{p+1}) = Reach_{G^p}(i,S_i^p)\bigcup\{l\in\chi^{p+1}\,|\,l>i\}\ .$$

(6) $i>\omega_q^{p+1}$, $i\in X^p$ but $i\notin\Delta$:

Row i of R^p is non-null and it is not used as a pivot row. Thus the structure of row i remains the same after the elimination of the rows of Z_{p+1}; that is, $R_{ij}^{p+1}=R_{ij}^p$, $j\geq i$. Now consider the graph G^{p+1}. Let s be a node in χ^{p+1} such that $s>\omega_q^{p+1}$. If there is a path joining s and i, the path must contain ω_q^{p+1}. However by Lemma 3.4.2, there must be a node $v>i$ on this path too. Hence $s\notin Reach_{G^{p+1}}(i,S_i^{p+1})$. That is,

$$Reach_{G^{p+1}}(i,S_i^{p+1}) = Reach_{G^p}(i,S_i^p)\ .$$

(7) $i>\omega_q^{p+1}$ and $i\in\chi^{p+1}$:

Assume $\delta_k<i<\delta_{k+1}$. Because row i of R^p is null and because of Lemma 2.1.5, the column indices of the off-diagonal nonzeros in row i of R^{p+1} are given by $\{j>i\,|\,R_{\delta_k j}^{p+1}\neq 0\}$. From (5) this is the same as

$$\{j\,|\,R_{\delta_k j}^p\neq 0\}\bigcup\{j\in\chi^{p+1}\,|\,j>i\}-\{\delta_k\}\ .$$

Note that in G^{p+1}, $i=\min\{l\,|\,l\in Reach_{G^{p+1}}(\delta_k,S_{\delta_k}^{p+1})\}$. Since $X^k\bigcap\chi^{k+1}=\{\omega_q^{p+1}\}$,

$$Reach_{G^{p+1}}(i,S_i^{p+1}) = Reach_{G^{p+1}}(\delta_k,S_{\delta_k}^{p+1})-\{i\}$$
$$= Reach_{G^p}(\delta_k,S_{\delta_k}^{p})\bigcup\{l\in\chi^{p+1}\mid l>\delta_k\}-\{i\} \quad .$$

Thus in each of the seven cases, $Reach_{G^{p+1}}(x,S_x^{p+1})$ is exactly the set of column indices in row x of R^{p+1}.

The proofs of cases (c), (d) and (e) are similar to that of case (b), and for this reason are omitted.

□

The fact that Lemma 3.3.1 predicts the structure of R correctly follows immediately from the previous result.

Corollary 3.4.4

Let A be the rectangular matrix associated with an n by n grid. Suppose R is the n^2 by n^2 upper triangular matrix obtained by reducing the rows of A using rotations. Let $G=(X,E)$ be the graph of A^TA. Then the nonzero structure of row i of R is given correctly by $Reach_G(i,S_i)$, where $S_i=\{l\mid l<i\}$.

□

A close examination of the elimination process reveals that if the rows of Z_p are eliminated together, the elimination sequence of the *fourth* row is always maximal.

The technique used in the proof of Theorem 3.4.3 can be used to prove a more general result which we state below.

Theorem 3.4.5

Let A be an M by N matrix. Partition the rows of A into $Z_1, Z_2, \cdots, Z_t$ so that *all* the rows of Z_i have the same structure. Suppose the following conditions hold.

(1) The number of rows in Z_i is greater than or equal to the number of nonzeros in each row of Z_i.

(2) All the rows of Z_i are eliminated together.

Let R be the N by N upper triangular matrix obtained in the orthogonal reduction of A. Then

(1) the elimination sequences of the last $|Z_i|-w$ rows of Z_i are maximal, where w is the number of nonzeros in each row of Z_i,

(2) Lemma 3.3.1 predicts correctly the structure of R from the graph of A^TA.

□

3.5. Planar graphs, finite element graphs and separator theorems

Certain problems defined on two-dimensional finite element graphs are examples that satisfy the conditions in Theorem 3.4.5. We refer to these problems as finite element problems. In fact, the model problem we introduced in the previous section is a special case of these problems. Before we describe the finite element problems, we define finite element graphs in terms of planar graphs.

A graph is said to be *planar* if it can be represented on a plane in such a way that its nodes correspond to *distinct points* on the plane, its edges correspond to *simple curves* on the plane, and no two curves share a point except possibly a common end-point. The corresponding graph on the plane is called a *planar embedding* of the planar graph. A *face* of a planar graph is a region of the planar embedding bounded by edges of the planar graph that does not contain any nodes or edges in its interior. Note that every planar graph has a number of *finite faces* and exactly one *infinite face*. In the following discussion, a "face" will refer to a finite face. The set of edges that surround a face is called its *boundary*. A *diagonal* of a face is an edge joining non-adjacent nodes on its boundary.

For our purpose, a *finite element graph* is a graph formed from a planar embedding of a planar graph by adding *all* possible diagonals to each face. Thus the nodes of each face of the planar embedding form a clique in the finite element graph. The planar graph or its planar embedding is called the *skeleton* of the finite element graph and each of its faces is an *element* of

the finite element graph.

A *finite element problem* is defined as follows. We assume that there is one variable associated with each node of a finite element graph. Suppose the i–th element has σ_i nodes. Then there are *at least* σ_i equations associated with this element and only the variables associated with the nodes of this element are involved in each equation.

Some of these finite element problems are important since their finite element graphs are guaranteed to have "small" width-1 and width-2 separators. As we see in Chapters 4 and 5, this property is important when we consider the problem of finding good row and column orderings.

We now review the *separator theorems* for planar and finite element graphs. Lipton and Tarjan recently prove that any planar graph with n nodes has a width-1 separator which has $O(\sqrt{n})$ nodes [57]. Furthermore, when this separator is removed, the graph is decomposed into two pieces, each of which contains no more than $\frac{2}{3}n$ nodes. This is summarized in Lemma 3.5.1. An $O(n)$-time algorithm for finding such a separator is also provided. One can use the result to construct an $O(n\log n)$-time algorithm to find a symmetric ordering for symmetric positive definite systems that are defined on planar graphs [40, 56, 57]. The number of nonzeros in the Cholesky factor of the reordered matrix is $O(n\log n)$.

Lemma 3.5.1

Let $G=(X,E)$ be a planar graph with n nodes. Then the nodes of X can be partitioned into three sets A, B and C such that

(1) if $x\in A$ and $y\in B$, then $d(x,y)>1$ (that is, C is a width-1 separator),

(2) neither A nor B contains more that $\frac{2}{3}n$ nodes, and

(3) C contains no more than $2\sqrt{2}\sqrt{n}$ nodes.

□

Unfortunately the existence of small width-1 separators in a planar graph does not necessarily imply the existence of small width-2 separators. Consider the planar graph in Figure 3.5.1. The set $\{x,y,z\}$ is a width-1 separator, but the graph *does not* have a small width-2 separator. The problem in this example is that the degree of y is not bounded. It is almost equal to the number of nodes in the graph. In fact, if the degree of any node is bounded, it can then be proved that the planar graph will have a width-2 separator that has $O(\sqrt{n})$ nodes.

Theorem 3.5.2

Let $G=(X,E)$ be a planar graph with $|X|=n$. Assume that each node is shared by no more that δ faces (that is, the degree of each node is bounded by $\delta+1$). Then the nodes of X can be partitioned into three sets A, B and C such that

(1) if $x\in A$ and $y\in B$, then $d(x,y)>2$ (that is, C is a width-2 separator),

(2) neither A nor B contains more that $\frac{2}{3}n$ nodes.

(3) C contains no more than $2\sqrt{2}(\delta+1)\sqrt{n}$ nodes.

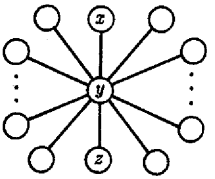


Figure 3.5.1 A planar graph that does not have a small width-2 separator.

Proof:

According to Theorem 3.5.1 the nodes of G can be partitioned into three sets $\overline{A}$, $\overline{B}$ and $\overline{C}$ such that $|\overline{A}| \leq \frac{2}{3}n$, $|\overline{B}| \leq \frac{2}{3}n$, $|\overline{C}| \leq 2\sqrt{2}\sqrt{n}$, and $\overline{C}$ is a width-1 separator.

Without loss of generality, assume $|\overline{A}| \geq |\overline{B}|$. Then define

$$A = \overline{A} - Adj(\overline{C}), \quad B = \overline{B}, \quad \text{and} \quad C = \overline{C} \bigcup \left(\overline{A} \bigcap Adj(\overline{C})\right) .$$

Clearly, because of the choice of C, C is a width-2 separator. Moreover we have

$$|A| = |\overline{A} - Adj(\overline{C})| \leq |\overline{A}| \leq \frac{2}{3}n ,$$

$$|B| = |\overline{B}| \leq \frac{2}{3}n ,$$

$$\begin{aligned} |C| &= |\overline{C} \bigcup \left(\overline{A} \bigcap Adj(\overline{C})\right)| = |\overline{C}| + |\overline{A} \bigcap Adj(\overline{C})| \\ &\leq 2\sqrt{2}\sqrt{n} + |Adj(\overline{C})| \\ &\leq 2\sqrt{2}\sqrt{n} + \delta|\overline{C}| \\ &\leq 2\sqrt{2}\sqrt{n} + 2\sqrt{2}\delta\sqrt{n} \\ &= 2\sqrt{2}(\delta+1)\sqrt{n} . \end{aligned}$$

□

The proof of Theorem 3.5.2 provides a way of constructing an algorithm for finding a width-2 separator in a planar graph. Note that the complexity of determining A, B and C from $\overline{A}$, $\overline{B}$ and $\overline{C}$ is $O(n)$ since at most n nodes are to be examined. Thus a width-2 separator can be found in $O(n)$ time.

Theorem 3.5.2 is similar to a result stated by Gilbert in [40]. He considers the use of width-2 separators in Gaussian elimination with partial pivoting for sparse matrices.

Suppose G is a finite element graph with n nodes. In [57], it is proved that G has a width-1 separator if each element has a bounded number of nodes.

Lemma 3.5.3

Let $G=(X,E)$ be a finite element graph with n nodes. Suppose no element of G has more than σ boundary nodes. Then the nodes of X can be partitioned into three sets A, B and C such that

(1) if $x \in A$ and $y \in B$, then $d(x,y)>1$ (that is, C is a width-1 separator),

(2) neither A nor B contains more that $\frac{2}{3}n$ nodes, and

(3) C contains no more than $4\left\lfloor\frac{\sigma}{2}\right\rfloor\sqrt{n}$ nodes.

□

As in the case of planar graphs, a finite element graph may not have a small width-2 separator even if it has a small width-1 separator. However if we impose certain restrictions on the finite element graphs, we can also prove that any such finite element graph will have a width-2 separator that has $O(\sqrt{n})$ nodes.

Now consider a class of finite element graphs satisfying the following conditions.

(1) Each element has at most σ boundary nodes.

(2) Each node is shared by at most δ elements.

(3) Both σ and δ are small and independent of n, where n is the number of nodes in the finite element graph.

It might seem that condition (3) is too restrictive. However, it is not: typical values for σ are 3 and 4, which correspond to triangular and quadrilateral elements respectively, and typical values for δ range from 4 to 10.

We now prove that any finite element graph belonging to this restricted class has a small width-2 separator.

Theorem 3.5.4

Let $G=(X,E)$ be a finite element graph satisfying the conditions above. Then the nodes of X can be partitioned into three sets A, B and C such that

(1) if $x \in A$ and $y \in B$, then $d(x,y)>2$, (that is, C is a width-2 separator),

(2) neither A nor B contains more than $\frac{2}{3}n$ nodes,

(3) C contains no more that $4\sigma\delta\left\lfloor\frac{\sigma}{2}\right\rfloor\sqrt{n}$ nodes.

Proof:

The nodes of G can be partitioned into three sets $\overline{A}$, $\overline{B}$, and $\overline{C}$ which satisfy Lemma 3.5.3. Without loss of generality, assume $|\overline{A}| \geq |\overline{B}|$. Construct a width-2 separator as follows.

Let C be initially empty. If any element in $G(\overline{A} \bigcup \overline{C})$ contains nodes of $\overline{C}$, then all the nodes of that element are included in C. Clearly C is a width-2 separator. Furthermore, since at most δ elements share a given node and each element has at most σ nodes, the total number of nodes that are included into C must be bounded by

$$|C| \leq \sigma\delta|\overline{C}| \leq 4\sigma\delta\left\lfloor\frac{\sigma}{2}\right\rfloor\sqrt{n} \quad .$$

Let A be the set of nodes of G that are in $\overline{A}$ but not in C, and let B be the set of nodes of G that are in $\overline{B}$ but not in C. Neither A nor B will contain more than $\frac{2}{3}n$ nodes because

$$|A| = |\overline{A}-C| \leq |\overline{A}| \leq \frac{2}{3}n \quad ,$$

and

$$|B| = |\overline{B}-C| \leq |\overline{B}| \leq \frac{2}{3}n \quad .$$

□

The bound on the size of the width-2 separator may be pessimistic, since it is unlikely that all the elements that share a node of $\overline{C}$ will be in $G(\overline{A} \bigcup \overline{C})$.

CHAPTER 4

WIDTH-TWO NESTED DISSECTION ORDERINGS

In this chapter we describe a heuristic algorithm for ordering the rows and columns of a sparse rectangular matrix A simultaneously. The algorithm works with the graph of A^TA and is motivated by some of the results that were presented in the previous chapter. Denote the reordered matrix by $\overline{A}$ and let R be the upper triangular matrix obtained in the orthogonal reduction of $\overline{A}$. We show that the number of nonzeros in R and the cost of computing R using rotations are respectively $O(n^2\log_2 n)$ and $O(n^3)$ when A is the rectangular matrix associated with an n by n grid. We describe an heuristic scheme for generating the orderings automatically. Numerical experiments are presented.

4.1. Width-2 nested dissection

In Section 3 of Chapter 3 we presented some basic results relating row and column orderings. We first review some of the important ones. Here x_i is the node having labelling i.

(1) The rotation of row a^k into R^{k-1} can be viewed as merging the row graph ϕ^k of a^k with the graph G^{k-1}, where G^{k-1} is the graph of $(A^{k-1})^TA^{k-1}$ (or the union of ϕ^i, $1\leq i\leq k-1$).

(2) Let $\Xi^k=\{\xi_1^k,\xi_2^k,\cdots,\xi_{\mu_k}^k\}$ be the elimination sequence of a^k. Then Ξ^k is related to the component $G^k(C_{\sigma_k}^k)$ of G^k that contains ϕ^k (see Theorem 3.3.8). More precisely, the set $\{x_s \mid s\in\Xi^k\}$ is a subset of $C_{\sigma_k}^k$. Note that the cost of eliminating a^k depends partly on μ_k, the length of the elimination sequence.

(3) The cost of eliminating a^k also depends in part on the labelling of the nodes of the component $G^k(C_{\sigma_k}^k)$ (see Corollary 3.3.10). Loosely speaking, it depends on the number of nodes of $C_{\sigma_k}^k$ whose labellings are greater than ξ_1^k.

These results provide us with a way of constructing "good" row and column orderings. In order to ensure that the μ_k, $1 \leq k \leq m$, are small, the row ordering should be selected so that the components, $G^k(C^k_{\sigma_k})$, $1 \leq k \leq m$, can be kept small. On the other hand, the column ordering should be chosen so that the number of nodes in $C^k_{\sigma_k}$ whose labellings are greater than ξ^k_1 is kept small. Of course, the column ordering should also be chosen so that the upper triangular matrix R is sparse. How can these be achieved? A simple heuristic way is described below.

For simplicity we use the following notation in our discussion.

(1) Let Ξ^k be the elimination sequence of row a^k. The corresponding node set of G^k, namely $\{x_s \mid s \in \Xi^k\}$, is denoted by Δ^k. We also call Δ^k the elimination sequence of a^k.

(2) If Z is a set of row graphs, then $\chi(Z)$ is the union of the node sets of the row graphs of Z. The corresponding rows of A are referred to as the *rows of* Z.

Consider the graph $G=(X,E)$ of A^TA. From Lemma 3.3.3, $G=\bigcup_{k=1}^{m} \phi^k$, where ϕ^k is the row graph of a^k. We assume that G is connected. (If G is disconnected, we simply apply the strategies described below to each component consecutively.) Suppose we can find a set of row graphs Z_3 such that if this set is removed from G, the remaining graph is decomposed into two or more components. Assume there are two components and denote them by G_1 and G_2 respectively. For $i=1,2$, let Z_i be the set of row graphs remaining in G_i. Note that the node set of G_i is exactly $\chi(Z_i)$. Now $Z_1 \bigcup Z_2 \bigcup Z_3$ is the set of all row graphs in G, and $\chi(Z_1) \bigcup \chi(Z_2) \bigcup \chi(Z_3) = X$. Because of the choice of Z_3, $\chi(Z_1) \bigcap \chi(Z_2) = \emptyset$, but $\chi(Z_3) \bigcap \chi(Z_i) \neq \emptyset$, for $i=1,2$. We now reorder the rows of A so that those corresponding to the row graphs of Z_3 occur *last*. In other words, the rows of Z_3 will be eliminated last.

What has this procedure achieved? First we note that G_1 and G_2 are disconnected unless the row graphs of Z_3 are merged with G_1 and G_2. That is, even if we have formed G_1 and G_2 (or equivalently, processed the rows of Z_1 and Z_2), the graph G will not be formed until the row graphs of Z_3 are processed. Thus when a row graph ϕ^k of either Z_1 or Z_2 is merged with an

existing graph G^{k-1}, the component that contains Δ^k is, in the worst case, G_1 or G_2 as long as the row graphs of Z_3 are processed last. This is a consequence of Theorem 3.3.8. That is, by eliminating the rows of Z_3 last, the length of the elimination sequence of any row of Z_1 or Z_2 is *limited to a proper subgraph of* G. It is important to note also that the elimination sequences of the rows of Z_1 and those of the rows of Z_2 are completely disjoint, regardless of the order in which these rows are processed. Thus the same technique can be employed to label the rows of Z_1 and Z_2 recursively.

Now assume that the rows of Z_1 and Z_2 have been eliminated; that is, G_1 and G_2 have been formed. Consider the elimination of the rows of Z_3. Let a^k be any row of Z_3 and Ξ^k be its elimination sequence. Theorem 3.3.8 says that Δ^k is contained in the component $G^k(C^k_{\sigma_k})$, where the node set $C^k_{\sigma_k}$ contains χ^k. Because of the row ordering strategy, this immediately means that the component $G^k(C^k_{\sigma_k})$ will include *both* G_1 and G_2. Hence the elimination sequence Δ^k of a^k *may* include nodes of G_1 and G_2. Thus, even though the lengths of the elimination sequences of the rows of Z_1 and Z_2 are limited, the elimination sequence of any row of Z_3 may be long, which is undesirable. An important observation is that relabelling the nodes of G (that is, reordering the columns of A) *does not* change the fact that G_1 and G_2 are disconnected (before the rows of Z_3 are processed). Yet it may be possible to limit the length of the elimination sequence of any row of Z_3. Consider the following strategy. Suppose we relabel the nodes of G so that the nodes of $\chi(Z_3)$ are labelled *last*. Then, because of Corollary 3.3.10, the elimination sequence Δ^k of any row of Z_3 contains only nodes of $\chi(Z_3)$, even though the component containing the corresponding row graph could be as large as G. Naturally we want to choose Z_3 so that $\chi(Z_3)$ is small, compared to the size of G.

Another important consequence of the node labelling strategy is the following. Denote the reordered matrix by $\overline{A}$ and let R be the upper triangular matrix in the upper trapezoidal form. Suppose $x_i \in \chi(Z_1) - \chi(Z_3)$ and $x_j \in \chi(Z_2) - \chi(Z_3)$, where x_i is the node having labelling i in the relabelled graph. Assume $i < j$. Then Lemmas 3.3.1 and 3.3.4 imply that $R_{ij} = 0$ since there is

no path joining x_i and x_j in G that involves nodes whose labellings are *less than both* i and j.

Thus our ordering strategies guarantee that

(1) the fill-in in R will be *low*,

(2) the rows of Z_3 will not be expensive to eliminate, since the elimination sequence of any row of Z_3 is limited to a subset of nodes of G that have the largest labellings, and

(3) the elimination sequence of any row of Z_1 or Z_2 is limited to a proper subgraph of G.

We now summarize our ordering strategy. Find a set of row graphs Z_3 such that

(1) $\chi(Z_3)$ is small, and

(2) removing the row graphs of Z_3 from G disconnects the remaining graph into two or more components.

Then we reorder the rows and columns of A so that

(1) the rows of Z_3 appear last, and

(2) the nodes of $\chi(Z_3)$ are labelled last.

Note that the first one implies a partial *row ordering* and the second one implies a partial *column ordering*. Thus our ordering technique determines *both* row and column orderings *simultaneously*. Figure 4.1.1 illustrates the structure induced in the reordered matrix by this process. For clarity, the rows of Z_1 and Z_2 have been arranged consecutively, and the nodes of $\chi(Z_1)$ and $\chi(Z_2)$, apart from those appearing in $\chi(Z_3)$, have also been arranged consecutively. The portion of R that is involved when the rows of Z_i, $1 \leq i \leq 3$, are eliminated is also shown.

The ordering technique also allows us to derive simple bounds on the number of nonzeros in R and the cost of computing R. Let $n_i = |\chi(Z_i)|$ and $m_i = |Z_i|$, $1 \leq i \leq 3$. Assuming the non-null blocks of $\overline{A}$ and R are full, the number of nonzeros in R is bounded by

$$\frac{1}{2}\sum_{i=1}^{3} n_i(n_i+1) \ ,$$

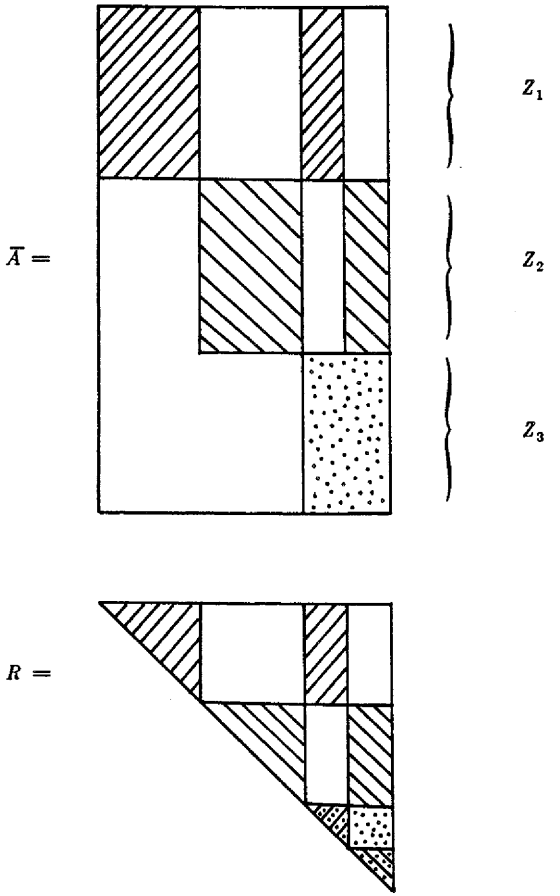


Figure 4.1.1 Structures of $\bar{A}$ and R induced by the row and column ordering strategies.

and the cost of computing R is bounded by

$$\frac{1}{2}\sum_{i=1}^{3} m_i n_i(n_i+1) \quad .$$

Note that $m_1+ m_2+ m_3=m$, but in general $n_1+ n_2+ n_3>n$.

Finally note that the technique has *decomposed* the problem into two smaller subproblems. The elimination of the rows of Z_1 and the elimination of the rows of Z_2 involve two completely independent subgraphs. Thus the same strategies can be recursively applied to the row graphs of Z_1 and Z_2 (or G_1 and G_2) respectively. This refinement can be repeated until no further decomposition is possible. Note that each level of refinement would allow the determination of better bounds on the number of nonzeros in R and the cost of computing R.

Clearly the effectiveness of our strategies depends on how the set of row graphs Z_3 can be identified. Since $m>>n$ in general, it may not be economical, both in terms of space and time, to work with the sequence of graphs G^k or the sequence of row graphs ϕ^k for determining Z_3. Recall that the graph G of A^TA is simply the union of all the row graphs and is the same as G^m. Hence it seems more natural to work with G. We now show that the set Z_3 can be identified easily if we can determine a minimal width-2 separator in the graph of A^TA.

For simplicity we assume $G=(X,E)$ is connected. Let S be a separator of G whose removal disconnects the graph into two or more components (say two). Denote the node sets of the two components by C_1 and C_2 respectively. The first result follows directly from the definition of a separator.

Lemma 4.1.1

Let K be any clique in G. Then either $K\subseteq C_1\bigcup S$ or $K\subseteq C_2\bigcup S$.

□

We now assume that S is a width-2 separator. Recall that each row graph $\phi^k=(\chi^k,\epsilon^k)$ which corresponds to row a^k of A is a complete graph. The following lemmas characterize these

row graphs in the partitioning $\{C_1, C_2, S\}$ of X.

Lemma 4.1.2

Assume S is a width-2 separator. Let $\chi^i \subseteq C_1 \bigcup S$ and $\chi^j \subseteq C_2 \bigcup S$, $i \neq j$. If $\chi^i \bigcap C_1 \neq \emptyset$ and $\chi^j \bigcap C_2 \neq \emptyset$, then $\chi^i \bigcap \chi^j = \emptyset$.

Proof:

If $\chi^i \bigcap S = \emptyset$ or $\chi^j \bigcap S = \emptyset$, then the result follows immediately from Lemma 4.1.1. Assume $\chi^i \bigcap S \neq \emptyset$ and $\chi^j \bigcap S \neq \emptyset$. Let $x_i \in \chi^i \bigcap C_1$ and $x_j \in \chi^j \bigcap C_2$. If $\chi^i \bigcap \chi^j \neq \emptyset$, there would exist $y \in \chi^i \bigcap \chi^j \subseteq S$ such that $x_i, x_j \in Adj(y)$, which contradicts the fact that S is a width-2 separator.

□

Lemma 4.1.3

If S is a width-2 separator, then there is at least one row graph ϕ^k in the section graph $G(S)$.

Proof:

Since S is a width-2 separator, there must exist $u \in C_1$ and $v \in C_2$ such that the distance between u and v in G is 3. Let the corresponding path joining u and v be (u, x, y, v), where $x, y \in S$. Now G is the union of the row graphs ϕ^i, $1 \leq i \leq m$. So there must exist one row graph, say $\phi^k = (\chi^k, \epsilon^k)$, such that $x, y \in \chi^k$ and $\{x, y\} \in \epsilon^k$. By Lemma 4.1.1, either $\chi^k \subseteq C_1 \bigcup S$ or $\chi^k \subseteq C_2 \bigcup S$. Assume the first alternative. If $\chi^k \bigcap C_1 \neq \emptyset$, then there exists $w \in \chi^k \bigcap C_1$ and a path (w, y, v) in G, contradicting the fact that S is a width-2 separator. Thus $\chi^k \subseteq S$ and ϕ^k is a subgraph of $G(S)$.

□

Lemma 4.1.4

If S is a minimal width-2 separator, then every edge in $G(S)$ belongs to at least one row graph ϕ^k of $G(S)$.

Proof:

This follows from the fact that G is a union of the row graphs ϕ^i and S is a minimal width-2 separator.

□

The fact that a minimal width-2 separator identifies a set of rows now follows from Lemmas 4.1.3 and 4.1.4.

Theorem 4.1.5

If S a minimal width-2 separator, then $G(S)$ is the union of one or more row graphs ϕ^k.

□

We now assume that S is a minimal width-2 separator. Theorem 4.1.5 provides us with a mechanism of identifying the sets of row graphs desired. Let Z_3 be the set of row graphs in $G(S)$. It follows from Lemma 4.1.3 that Z_3 must be non-null. Then Lemmas 4.1.1 and 4.1.2 imply that the node set of any row graph *not* in Z_3 must be contained in either $C_1 \bigcup Adj(C_1)$ or $C_2 \bigcup Adj(C_2)$. Note that $Adj(C_1)$, $Adj(C_2) \subset S$, and $Adj(C_1) \bigcap Adj(C_2) = \emptyset$ because S is a width-2 separator. For $i=1,2$, let Z_i be the set of row graphs in the section subgraph $G(C_i \bigcup Adj(C_i))$. Then $\chi(Z_i)$ is the node set of the union of the row graphs of Z_i. Since $\chi(Z_i) \subseteq C_i \bigcup Adj(C_i)$ for $i=1,2$, $\chi(Z_1)$ and $\chi(Z_2)$ are disjoint. Thus a minimal width-2 separator S identifies the set of rows and the set of nodes desired. Now we simply reorder the rows and columns of A so that

(1) the rows of Z_3 appear last, and

(2) the columns corresponding to the nodes in $\chi(Z_3)$ (that is, the nodes of S) are labelled last.

Since we intend to work with the graph G, determining the separator S does not immediately identify the set of row graphs Z_3. Fortunately this can be done cheaply, as the following discussion shows. We assume that the nodes of G have been relabelled so that the labellings of the nodes of S are *greater* than those of the nodes of C_1 and C_2. Define the *leading subscript* of a row to be the column index of the first nonzero in that row. In terms of graphs, the leading subscript of a row graph is the labelling of the node having the smallest labelling. Because of the labelling strategy for G, the labelling of any node in C_i, i=1,2, must be *less than* the labelling of any node in S. Thus if ϕ^s and ϕ^t are row graphs in Z_3 and Z_i respectively (i=1,2), then the leading subscript of ϕ^s (or a^s) must be *greater than* that of ϕ^t (or a^t). Thus the row ordering can be obtained simply by arranging the rows in (the column-permuted) A so that *the leading subscripts are in ascending order*. Note that it is assumed that the graph of A^TA (and its subgraphs) has width-2 separators.

This *dissection* technique can of course be applied recursively to the subgraphs $G(C_i)$, yielding a *width-2 nested dissection*, which is similar to the nested dissection described in [28]. However, care has to be taken when choosing width-2 separators. Consider the graph G=(X,E) in Figure 4.1.2.

Let S_1 be the set of nodes that are darkened. Clearly S_1 is a minimal width-2 separator of G. Denote the components of $G(X-S)$ by $G(C_1^1)$ and $G(C_2^1)$. Now we apply the dissection technique again to the graph $G(C_1^1)$ and find a minimal width-2 separator $\overline{S}_2$ ($\overline{S}_2$ contains nodes that are shaded). Denote the components in $G(C_1^1-\overline{S}_2)$ by $G(C_1^2)$ and $G(C_2^2)$. Note that even though $\overline{S}_2$ is a minimal width-2 separator of $G(C_1^1)$, $S_1 \bigcup \overline{S}_2$ is *not* a minimal width-2 separator of G with respect to the components $G(C_1^2)$, $G(C_2^2)$ and $G(C_2^1)$, since there is a path of length 2 from x to w. The problem is due to the fact that $Adj(C_1^1) \subset S_1$. In this case, we cannot guarantee that there would not exist $u \in C_1^2$ and $v \in C_2^2$ such that $S_1 \bigcap Adj(u) \bigcap Adj(v) \neq \emptyset$.

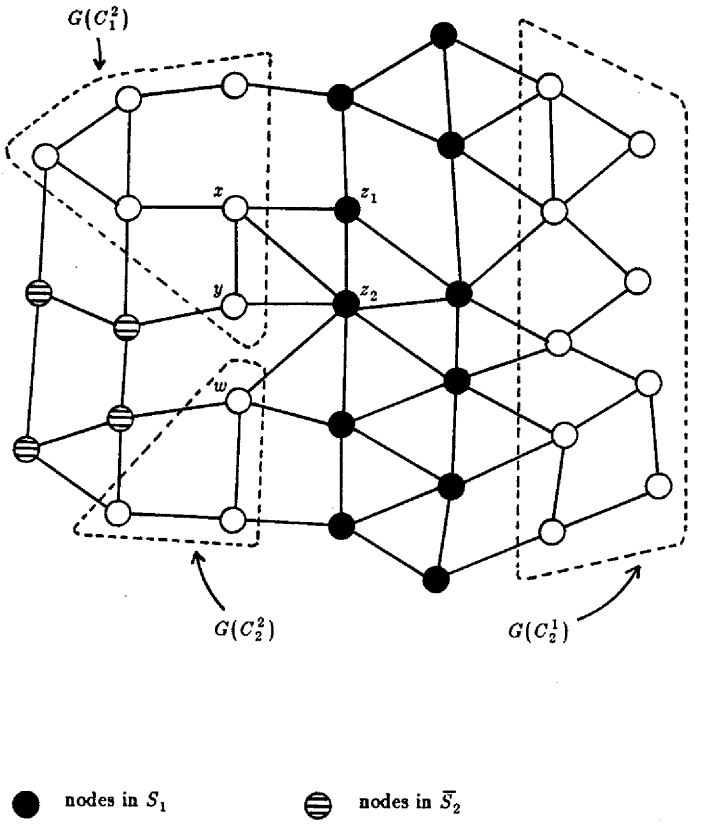


Figure 4.1.2 An example illustrating the choice of width-2 separators.

To solve this problem, we proceed as follows. Instead of choosing a width-2 separator from $G(C_1^1)$, we choose it from the graph $G(C_1^1 \bigcup Adj(C_1^1))$. As an example, $S_2=\overline{S}_2 \bigcup \{x,y,z_1,z_2\}$ is a minimal width-2 separator of $G(C_1^1 \bigcup Adj(C_1^1))$ in Figure 4.1.2, and $S_1 \bigcup S_2=S_1 \bigcup \overline{S}_2 \bigcup \{x,y\}$ is a minimal width-2 separator of G. The following theorem shows that the separator constructed in this way is always a minimal width-2 separator of G.

Theorem 4.1.6

Let S_1^1 be a minimal width-2 separator of a connected graph $G=(X,E)$ and denote the components in $G(X-S_1^1)$ by $G(C_1^1)$, $G(C_2^1)$, $\cdots$, $G(C_p^1)$. Let S_k^2 be a minimal width-2 separator of $G(C_k^1 \bigcup Adj(C_k^1))$ and denote the components in $G(C_k^1-S_k^2)$ by $G(C_1^2)$, $G(C_2^2)$, $\cdots$, $G(C_q^2)$. If $x \in C_i^2$ and $y \in C_j^2$, $i \neq j$, then the distance between x and y in G is greater than 2. That is, $S_1^1 \bigcup S_k^2$ is a width-2 separator of G. Furthermore, $S_1^1 \bigcup S_k^2$ is minimal.

Proof:

If $x \in C_i^2$ and $y \in C_j^2$, $i \neq j$, then $Adj(x) \subseteq C_i^2 \bigcup Adj(C_i^2)$ and $Adj(y) \subseteq C_j^2 \bigcup Adj(C_j^2)$. Note that $G(C_i^2 \bigcup Adj(C_i^2)-S_k^2)$ and $G(C_j^2 \bigcup Adj(C_j^2)-S_k^2)$ are components in $G(C_k^1 \bigcup Adj(C_k^1))$, and S_k^2 is a minimal width-2 separator of $G(C_k^1 \bigcup Adj(C_k^1))$. Thus, $C_i^2 \bigcup Adj(C_i^2)$ and $C_j^2 \bigcup Adj(C_j^2)$ must be disjoint, implying that $Adj(x) \bigcap Adj(y)=\emptyset$. Now $S_1^1 \bigcup S_k^2$ is minimal because S_1^1 and S_k^2 are minimal, and G is a union of complete graphs.

□

We now define a *width-2 nested dissection partitioning* formally. Let $G=(X,E)$ be the unlabelled graph of A^TA. Let $Y^0=X$, and for $m=0,1,2,\cdots,h$ until $Y^{h+1}=\emptyset$, do the following:

(1) Determine the connected components of Y^m and label them Y_1^m, Y_2^m, $\cdots$, $Y_{r_m}^m$.

(2) For $j=1,2,\cdots,r_m$, choose $\overline{S}_j^m \subseteq Y_j^m \bigcup Adj(Y_j^m)$ such that $\overline{S}_j^m$ is a minimal width-2 separator of $G(Y_j^m \bigcup Adj(Y_j^m))$. If $\overline{S}_j^m \neq \emptyset$, set $S_j^m=\overline{S}_j^m \bigcap Y_j^m$. Otherwise, set $S_j^m=Y_j^m$.

(3) Define $S^m = \bigcup_{j=1}^{r_m} S_j^m$ and $Y^{m+1} = Y^m - S^m$.

The partitioning $\Phi = \{S_j^m \subseteq X, 1 \leq j \leq r_m, 0 \leq m \leq h\}$ is called a width-2 nested dissection partitioning of G.

An ordering of X is said to be a *width-2 nested dissection ordering* with respect to $\Phi = \{S_j^m\}$ if for $x \in S_j^m$ and $y \in Y_j^m - S_j^m$, x is labelled *after* y. An example of a width-2 nested dissection ordering with respect to the partitioning of Figure 4.1.3 is shown in Figure 4.1.4.

The following theorem is a minor modification of Theorem 2.4 in [28]. It provides a bound on the number of nonzeros in the upper triangular matrix R.

Theorem 4.1.7

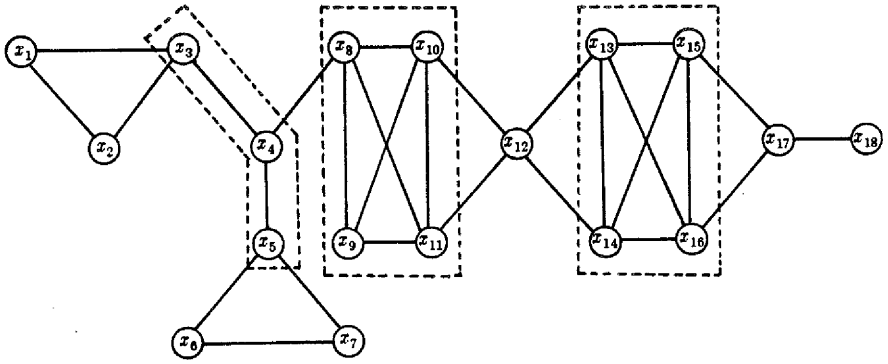
Let $\Phi = \{S_j^m\}$ be a width-2 nested dissection partitioning on G and the ordering of X be any width-2 nested dissection ordering with respect to Φ. Then the number of off-diagonal nonzeros in R is bounded by

$$\sum_{m=0}^{h} \sum_{j=1}^{r_m} |S_j^m| \left\{ |Adj(Y_j^m)| + \frac{(|S_j^m| - 1)}{2} \right\} .$$

□

This result is not only true for width-1 and width-2 nested dissection partitionings and orderings. It can be generalized for any width-l nested dissection ordering with $l \geq 1$.

We now derive the cost of computing R, where cost is measured as described in Chapter 2. The results in this section show that the node set S_j^m (a minimal width-2 separator of Y_j^m) identifies a set of rows of A, which we denote by Z_j^m. Let a^q be one of those rows, and consider the corresponding component partitioning ψ^q of Theorem 3.3.8. Obviously, one of its members contains the set $S_j^m \bigcup Adj(Y_j^m)$, and this set contains $\chi(Z_j^m)$, where $\chi(Z_j^m)$ is the union of the node sets of the row graphs of Z_j^m. Moreover by Corollary 3.3.10, the length of the elimination sequence of a^q will be bounded by μ_j^m, where



$$Y_1^0 = X \qquad S_1^0 = \{x_8, x_9, x_{10}, x_{11}\}$$

$$Y_1^1 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \qquad S_1^1 = \{x_3, x_4, x_5\}$$

$$Y_2^1 = \{x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}\} \qquad S_2^1 = \{x_{13}, x_{14}, x_{15}, x_{16}\}$$

$$Y_1^2 = S_1^2 = \{x_1, x_2\}$$

$$Y_2^2 = S_2^2 = \{x_6, x_7\}$$

$$Y_3^2 = S_3^2 = \{x_{12}\}$$

$$Y_4^2 = S_4^2 = \{x_{17}, x_{18}\}$$

Figure 4.1.3 A width-2 nested dissection partitioning $\{S_j^m\}$ of the graph of $A^T A$.

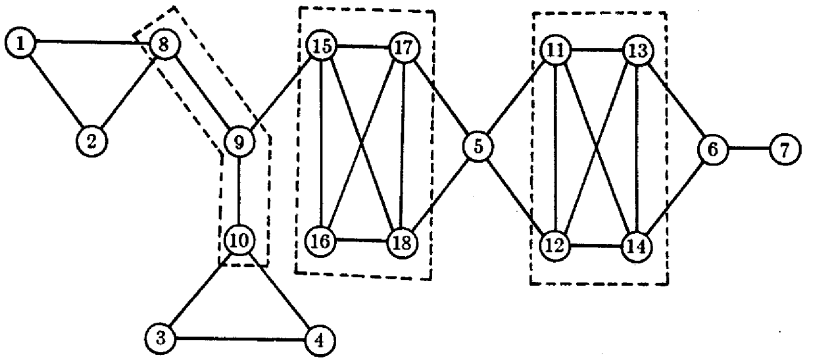


Figure 4.1.4 A width-2 nested dissection ordering on the width-2 nested dissection partitioning of Figure 4.1.3.

$$\mu_j^m = |\, S_j^m \bigcup Adj(Y_j^m) \,| \quad .$$

Thus we have the following theorem.

Theorem 4.1.8

Let $\Phi = \{S_j^m\}$ be a width-2 nested dissection partitioning on G and the ordering of X be any width-2 nested dissection ordering with respect to Φ. If the row ordering induced by the width-2 nested dissection ordering is used, then the cost of computing R is bounded by

$$\sum_{m=0}^{h} \sum_{j=1}^{r_m} \frac{|\, Z_j^m \,|\, \mu_j^m (\mu_j^m + 1)}{2} \quad .$$

□

Since both the number of nonzeros in R and the cost of computing R depend in part on the sizes of the width-2 separators, it is desirable to use small separators.

4.2. Complexity of width-2 nested dissection for the model problem

Consider the model problem introduced in Section 3.4. Let A be the rectangular matrix associated with an n by n grid. Denote the graph of A^TA by $G=(X,E)$. (Recall that G is almost identical to the grid.) We assume that the labelling of the nodes of G (and hence the column ordering of A) is a width-2 nested dissection ordering. An example is shown in Figure 4.2.1 with $n=14$. For clarity we have displayed the grid instead of the graph G. We also assume that the row ordering induced by width-2 nested dissection is used. That is, the rows are arranged so that the leading subscripts are in ascending order. Let R be the upper triangular matrix obtained in the orthogonal decomposition of A. In this section we investigate the quality of width-2 nested dissection orderings by deriving the number of nonzeros in R and the cost of computing R.

The technique used in the derivation is due to Rose and Whitten [66]. It involves the use of a so-called *bordered n by n grid* which is an n by n grid where one or more sides of this grid are bordered by an *additional* grid-line. This additional grid-line is sometimes referred to as a *boundary line*. Some examples of bordered 3 by 3 grids are shown in Figure 4.2.2. Note that when a grid is bordered along two sides, we assume that the two boundary lines are *adjacent* to each other (see the example in Figure 4.2.2). For our purpose the labellings of the grid points (or nodes) on the boundary lines are *always greater than* those of the nodes in the n by n grid. If R_i is an upper triangular matrix associated with an n by n grid which is bordered along i sides, then $\pi(n,i)$ and $\theta(n,i)$ will denote respectively the number of off-diagonal nonzeros in R_i and the cost of computing R_i. Hence $\pi(n,0)$ and $\theta(n,0)$ are the quantities we wish to determine. For convenience we assume $n=2^m-2$, for some integer $m>1$.

First we review the dissection strategy. Consider a p by q grid, with $p \leq q$. Any two consecutive (horizontal or vertical) grid-lines in its interior form a minimal width-2 separator.

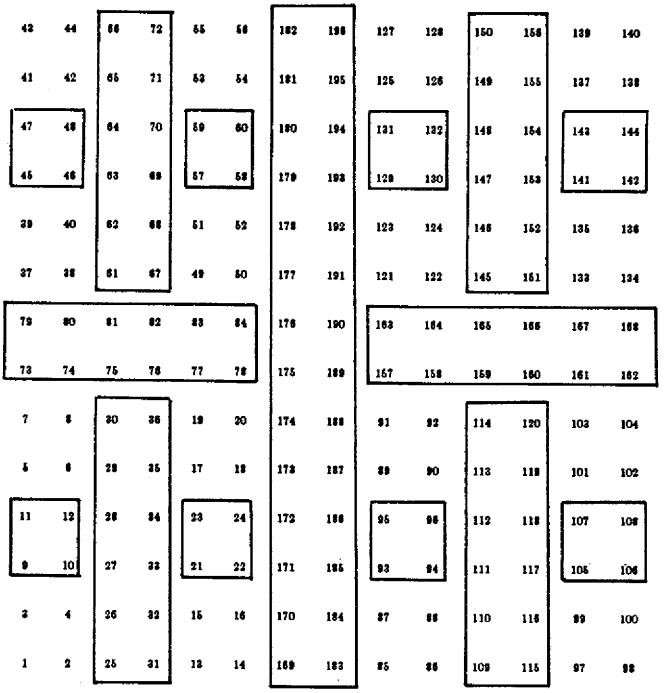


Figure 4.2.1 A width-2 nested dissection ordering on an 14 by 14 grid.

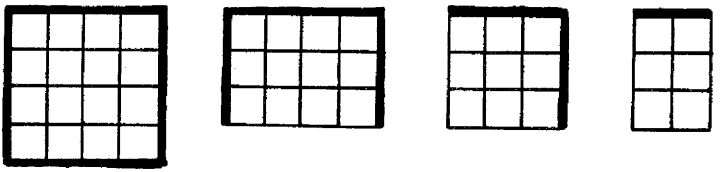


Figure 4.2.2 Examples of bordered 3 by 3 grids.

That is, a width-2 separator is either a p by 2 or q by 2 subgrid in the interior of the p by q grid. The discussion in the previous section indicated that the separator should be small. Thus we will use a p by 2 subgrid as a separator. Furthermore, for convenience, the p by 2 subgrid will be chosen so that removing this subgrid decomposes the p by q grid into two components, each of which is roughly a $\frac{p}{2}$ by q grid (see the example in Figure 4.2.1). In the following discussion, a *separator-line* refers to a grid-line in the separator with p nodes. Thus a width-2 separator consists of two parallel separator-lines. We assume that the nodes in a width-2 separator are labelled by separator-line (see Figure 4.2.1). The labellings of the nodes on one separator-line (called the *first separator-line*) will be less than those of the nodes on the other one (called the *second separator-line*).

Now consider an unbordered n by n grid. We first find a width-2 separator S_1 to dissect the grid into two n by $\frac{1}{2}(n-2)$ grids. Then for each n by $\frac{1}{2}(n-2)$ grid, we find a width-2 separator S_2 to dissect it into two $\frac{1}{2}(n-2)$ by $\frac{1}{2}(n-2)$ grids. Note that $|S_1|=2n$ and $|S_2|=n-2$. Hence after the dissection technique is applied twice, the n by n grid is dissected into four identical pieces, C_1, C_2, C_3 and C_4, each of which is an $\frac{1}{2}(n-2)$ by $\frac{1}{2}(n-2)$ grid

bordered along two sides. This is illustrated by an example in Figure 4.2.3. The observations below follow immediately from the ordering strategies.

(1) Let x_i denote the grid point (or node) having labelling i. If $x_i \in S_1$, $x_j \in S_2$, and $x_k \in C_l$, $1 \leq l \leq 4$, then $i > j > k$.

(2) The rows associated with the small squares in $C_l \bigcup Adj(C_l)$, $1 \leq l \leq 4$, will be processed first. Those in S_2 will be processed next, and those in S_1 are processed last.

The same dissection technique can be applied recursively to relabel the remaining nodes of C_l, $1 \leq l \leq 4$.

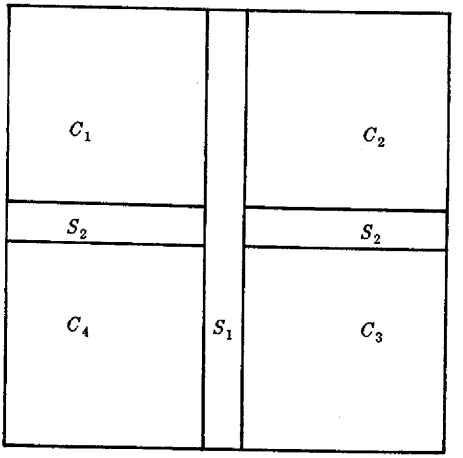


Figure 4.2.3 Structure of the grid after dissection technique is applied twice.

Let $Reach_G(x_i)$ denote the set $Reach_G(x_i,T_i)$, where $T_i=\{x_j\in X \mid j<i\}$. We will use $G_l=(X_l,E_l)$ to denote the graph of the subgrid containing $C_l \bigcup Adj(C_l)$, $1\leq l\leq 4$. The first and second separator-lines in S_i $(i=1,2)$ are denoted by S_{i1} and S_{i2} respectively. Let $\eta_{ij}+1$ denote the smallest labelling in S_{ij}.

The following observations will be useful in the derivation of $\pi(n,0)$.

(1) By Lemma 3.3.1 and Corollary 3.4.4, the number of off-diagonal nonzeros in R is

$$\pi(n,0) = \sum_{x\in X} \left| Reach_G(x) \right| \quad .$$

(2) For any $x\in C_l$, $1\leq l\leq 4$, $Reach_G(x)=Reach_{G_l}(x)$. This follows from the fact that the nodes of S_1 and S_2 are labelled last.

(3) The graphs G_1, G_2, G_3 and G_4 are identical. The nodes of each C_l will be assumed to be labelled in exactly the same way. Thus

$$\sum_{x\in C_p} \left| Reach_{G_p}(x) \right| = \sum_{x\in C_q} \left| Reach_{G_q}(x) \right| \quad , \quad 1\leq p,q\leq 4 \quad .$$

Note that G_p is simply the graph of an $\frac{1}{2}(n-2)$ by $\frac{1}{2}(n-2)$ grid which is bordered along two sides. Thus

$$\sum_{x\in C_p} \left| Reach_{G_p}(x) \right| = \pi(\frac{n-2}{2},2) \quad .$$

(4) Consider the two width-2 separators S_2 in Figure 4.2.3. Denote the one on the left by S_2^l and the one on the right by S_2^r. Assume the same dissection strategy is applied to each of the two n by $\frac{1}{2}(n-2)$ grids. Then

$$\sum_{x\in S_2^l} \left| Reach_G(x) \right| = \sum_{x\in S_2^r} \left| Reach_G(x) \right| \quad .$$

(5) Let $x\in S_{21}$. If the labelling of x is $\eta_{21}+j$, $1\leq j\leq \frac{1}{2}(n-2)$, then it can be shown that

$$|Reach_G(x)| = n \ .$$

This is illustrated in Figure 4.2.4. Darkened nodes are in $Reach_G(x) \bigcup \{x\}$. Nodes which are shaded have labels less than that of x.

(6) Let $x \in S_{22}$. If the labelling of x is $\eta_{22} + j$, $1 \leq j \leq \frac{1}{2}(n-2)$, then

$$|Reach_G(x)| = \frac{3}{2}n - 1 - j \ .$$

This is also illustrated in Figure 4.2.4.

(7) Using observations (5) and (6), we obtain the following.

$$\sum_{x \in S_2} |Reach_G(x)| = \sum_{x \in S_{21}} |Reach_G(x)| + \sum_{x \in S_{22}} |Reach_G(x)|$$

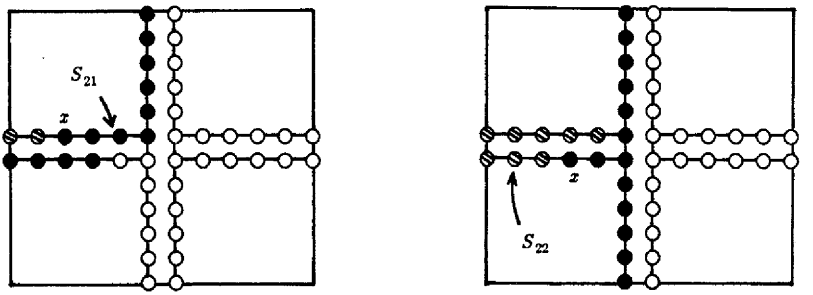


Figure 4.2.4 Darkened nodes are in $Reach_G(x) \bigcup \{x\}$.
Shaded nodes have labels less than that of x.

$$= \sum_{j=1}^{\frac{1}{2}(n-2)} n + \sum_{j=1}^{\frac{1}{2}(n-2)} (\frac{3}{2}n - 1 - j)$$

$$= \frac{9}{8}n^2 - \frac{11}{4}n + 1 \ .$$

(8) Let $x \in S_{11}$. If the labelling of x is $\eta_{11} + j$, $1 \leq j \leq n$, then

$$|Reach_G(x)| = n + 1 \ .$$

(When the labelling of x is $\eta_{11} + n$, $|Reach_G(x)| = n$.) This is illustrated in Figure 4.2.5. Nodes which are in $Reach_G(x) \bigcup \{x\}$ are darkened, and nodes whose labellings are less than that of x are shaded.

(9) Let $x \in S_{12}$. If the labelling of x is $\eta_{12} + j$, $1 \leq j \leq n$, then

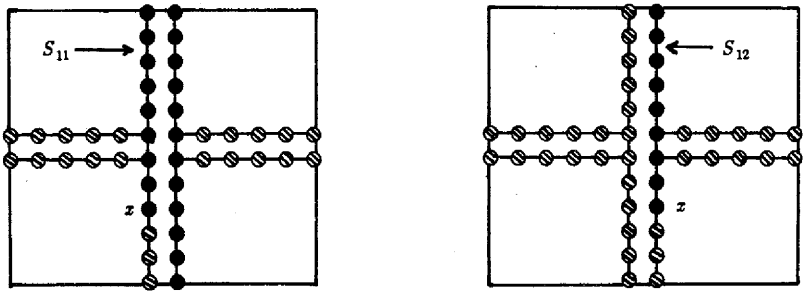


Figure 4.2.5 Same as Figure 4.2.4, except that $x \in S_1$.

$$|Reach_G(x)| = n-j \ .$$

This is also illustrated in Figure 4.2.5.

(10) Using observations (8) and (9), we obtain the following bound.

$$\sum_{x\in S_1} |Reach_G(x)| = \sum_{x\in S_{11}} |Reach_G(x)| + \sum_{x\in S_{12}} |Reach_G(x)|$$

$$= \sum_{j=1}^{n}(n+1) + \sum_{j=1}^{n}(n-j)$$

$$= \frac{3}{2}n^2 - \frac{1}{2}n \ .$$

Now we return to the derivation of $\pi(n,0)$. The number of off-diagonal nonzeros in $R=R_0$ is given by

$$\pi(n,0) = \sum_{x\in X} |Reach_G(x)| \ .$$

This can be simplified using the observations above:

$$\pi(n,0) = \sum_{k=1}^{4}\sum_{x\in C_k} |Reach_G(x)| + \sum_{x\in S_2^l} |Reach_G(x)| + \sum_{x\in S_2^r} |Reach_G(x)|$$

$$+ \sum_{x\in S_1} |Reach_G(x)|$$

$$= 4\pi(\frac{n-2}{2},2) + 2(\frac{9}{8}n^2 - \frac{11}{4}n + 1) + (\frac{3}{2}n^2 - \frac{1}{2}n)$$

$$= 4\pi(\frac{n-2}{2},2) + \frac{15}{4}n^2 - 6n + 2 \ . \qquad (4.2.1)$$

Using the same technique and arguments, we can derive the recurrence equations for $\pi(n,i)$, $i=2,3,4$, which are given in (4.2.2)-(4.2.4). The choices of width-2 separators in various bordered n by n grids are shown in Figure 4.2.6.

$$\pi(n,2) = \pi(\frac{n-2}{2},2) + 2\pi(\frac{n-2}{2},3) + \pi(\frac{n-2}{2},4) + \frac{15}{2}n^2 - 6n + 1 \ , \qquad (4.2.2)$$

$$\pi(n,3) = 2\pi(\frac{n-2}{2},3) + 2\pi(\frac{n-2}{2},4) + \frac{41}{4}n^2 - 5n \ , \qquad (4.2.3)$$

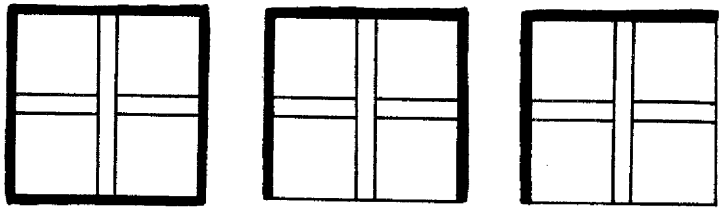


Figure 4.2.6 Choice of width-2 separators in various bordered n by n grids.

$$\pi(n,4) = 4\pi(\frac{n-2}{2},4) + \frac{51}{4}n^2 - 2n - 4 \ . \tag{4.2.4}$$

Since $n=2^m-2$, we have $\frac{n-2}{2}=2^{m-1}-2$. Thus equations (4.2.1)-(4.2.4) can be restated in terms of m:

$$\pi(m,0) = 4\pi(m-1,2) + \frac{15}{4}2^{2m} - 21(2^m) + 29 \ , \tag{4.2.5}$$

$$\pi(m,2) = \pi(m-1,2) + 2\pi(m-1,3) + \pi(m-1,4) + \frac{15}{2}2^{2m} - 36(2^m) + 43 \ , \tag{4.2.6}$$

$$\pi(m,3) = 2\pi(m-1,3) + 2\pi(m-1,4) + \frac{41}{4}2^{2m} - 46(2^m) + 51 \ , \tag{4.2.7}$$

$$\pi(m,4) = 4\pi(m-1,4) + \frac{51}{4}2^{2m} - 53(2^m) + 51 \ . \tag{4.2.8}$$

The following lemma is useful when we determine a closed form for $\pi(m,0)$. It can be proved by induction.

Lemma 4.2.1

(a) If $f(m) = f(m-1) + \alpha m 2^{2m} + O(2^{2m})$, then $f(m) = \frac{4}{3}\alpha m 2^{2m} + O(2^{2m})$.

(b) If $g(m) = 2g(m-1) + \alpha m 2^{2m} + O(2^{2m})$, then $g(m) = 2\alpha m 2^{2m} + O(2^{2m})$.

(c) If $h(m) = 4g(m-1) + \alpha 2^{2m} + O(2^{m})$, then $h(m) = \alpha m 2^{2m} + O(2^{2m})$.

□

Applying Lemma 4.2.1 (c) to (4.2.8), we have

$$\pi(m,4) = 4\pi(m-1,4) + \frac{51}{4}2^{2m} + O(2^{m}) = \frac{51}{4}m2^{2m} + O(2^{2m}) \quad . \tag{4.2.9}$$

Using Lemma 4.2.1 (b) and (4.2.9), (4.2.7) becomes

$$\pi(m,3) = 2\pi(m-1,3) + \frac{51}{8}m2^{2m} + O(2^{2m}) = \frac{51}{4}m2^{2m} + O(2^{2m}) \quad . \tag{4.2.10}$$

Now using Lemma 4.2.1 (a), (4.2.9) and (4.2.10), (4.2.6) can be written as

$$\pi(m,2) = \pi(m-1,2) + \frac{153}{16}m2^{2m} + O(2^{2m}) = \frac{51}{4}m2^{2m} + O(2^{2m}) \quad . \tag{4.2.11}$$

Finally the closed form for $\pi(m,0)$ is obtained from (4.2.5), (4.2.11) and Lemma 4.2.1 (c):

$$\pi(m,0) = 4\pi(m-1,2) + \frac{15}{4}2^{2m} + O(2^{m}) = \frac{51}{4}m2^{2m} + O(2^{2m}) \quad . \tag{4.2.12}$$

Replacing m and 2^m by $\log_2(n+2)$ and $(n+2)$ respectively and expanding $\log_2(n+2)$, we have

$$\pi(n,0) = \frac{51}{4}n^2\log_2 n + O(n^2) \quad . \tag{4.2.13}$$

Thus the number of nonzeros in the upper triangular matrix R is $O(n^2\log_2 n)$.

It should be pointed out that the bounds in (4.2.1)-(4.2.4) are obtained by assuming a particular labelling of the nodes of S_1 and S_2. Such an assumption will make the derivation of $\theta(n,i)$ much simpler. It is possible to derive the worst case behavior without assuming any particular ordering. For example, let η_1+1 and η_2+1 be the smallest labellings in S_1 and S_2 respectively. Let $x \in S_1$ and $y \in S_2$. If the labellings of x and y are respectively η_1+i and η_2+j,

then, in the worst case, we have

$$|Reach_G(x)| = 2n - i$$

and

$$|Reach_G(y)| = 2n - 2 - j \quad .$$

That is, the submatrices corresponding to the nodes of S_1 and S_2 are full matrices. These bounds are attainable by treating a separator as a p by 2 grid and labelling the nodes *row by row*. Thus

$$\sum_{x \in S_1} |Reach_G(x)| = \sum_{i=1}^{2n} (2n - i) = 2n^2 - n$$

and

$$\sum_{y \in S_2} |Reach_G(y)| = \sum_{j=1}^{n-2} (2n - 2 - j) = \frac{3}{2}n^2 - \frac{9}{2}n + 3 \quad .$$

Hence the upper bound on $\pi(n,0)$ is given by

$$\begin{aligned} \pi(n,0) &= \sum_{x \in X} |Reach_G(x)| \\ &= 4\pi(\frac{n-2}{2},2) + 2(\frac{3}{2}n^2 - \frac{9}{2}n + 3) + (2n^2 - n) \\ &= 4\pi(\frac{n-2}{2},2) + 5n^2 + O(n) \quad . \end{aligned}$$

Similarly it is possible to derive the worst case situations for $\pi(n,i)$, i=2,3,4.

$$\pi(n,2) = \pi(\frac{n-2}{2},2) + 2\pi(\frac{n-2}{2},3) + \pi(\frac{n-2}{2},4) + 11n^2 + O(n) \quad ,$$

$$\pi(n,3) = 2\pi(\frac{n-2}{2},3) + 2\pi(\frac{n-2}{2},4) + 14n^2 + O(n) \quad ,$$

$$\pi(n,4) = 4\pi(\frac{n-2}{2},4) + 17n^2 + O(n) \quad .$$

Using the previous technique, it is then possible to show that, in *the worst case*,

$$\pi(n,0) = 17n^2 \log_2 n + O(n^2) \quad . \qquad (4.2.14)$$

The leading terms in both (4.2.13) and (4.2.14) are $O(n^2\log_2 n)$, but the coefficient in (4.2.14) is slightly larger than that in (4.2.13).

It is interesting to note the following. Consider an n by n grid. Let x_j denote the node having labelling j. Define a symmetric matrix B as follows: for $j>i$, $B_{ij}\neq 0$ if x_i and x_j are associated with the same small square. The unlabelled graph of B is *exactly* the same as the unlabelled graph G of A^TA. Moreover the structure of R is the same as that of the Cholesky factor L of B if the rows and columns of B are reordered using the width-2 nested dissection. In [52] it is proved that the number of nonzeros in L, for *any* symmetric ordering, is at least $O(n^2\log_2 n)$. Thus we can immediately conclude that, in terms of nonzero count, width-2 nested dissection is optimal (in the order of magnitude sense).

To derive the cost of computing R, we first make a few observations. Here $\hat{G}=(\hat{X},\hat{E})$ will denote the union of the row graphs of those rows that have been eliminated.

(1) Let $\hat{\mathbb{S}}$ be the elimination sequence of the current row. Then the cost of eliminating this current row using Algorithm 2.1.1 is

$$\hat{\alpha} = \sum_{s\in\hat{\mathbb{S}}} \{ |\, Reach_{\hat{G}}(x_s,\hat{S}_s)\,| + 1\} \ ,$$

where x_s is the node having labelling s and $\hat{S}_s=\{x_i\in\hat{X}\,|\,i<s\}$ (see Chapter 2 for details). Thus the total cost of eliminating the rows of A is

$$\sum_k \alpha^k \ .$$

(2) For *any* node labelling, the leading subscripts of the four rows associated with a small square are the same. Thus if the row ordering induced by width-2 nested dissection is used, it is reasonable to assume that those four rows will be eliminated together. Furthermore, because of the discussion in Section 3.4, the elimination sequence of the fourth row is maximal. The elimination sequences of the first three rows are subsets of that of the fourth one.

(3) If we assume that the nodes of C_l, $1 \leq l \leq 4$, are labelled in the same manner, then the costs of eliminating the rows of G_l, $1 \leq l \leq 4$, are identical. Moreover, since G_l is bordered along two sides, the cost of eliminating the rows of G_l is simply $\theta(\frac{n-2}{2}, 2)$.

(4) Similarly we assume that the cost of eliminating the rows in $G(S_2^l)$ is the same as the cost of eliminating the rows in $G(S_2^r)$.

(5) The row ordering induced by width-2 nested dissection implies that, *before* the rows of $G(S_1)$ and $G(S_2)$ are eliminated, the rows of G_l, $1 \leq l \leq 4$, must have been eliminated and G_l, $1 \leq l \leq 4$, are disjoint.

(6) Consider any row in $G(S_2)$ and assume its elimination sequence is maximal. Because of the way in which the nodes are labelled, the node which corresponds to the leading subscript must be in S_{21}. Let the leading subscript be $\eta_{21}+j$, $1 \leq j \leq \frac{n-2}{2}$. Using Lemma 3.3.5, the elimination sequence is

$$\{\eta_{21}+i \mid j \leq i \leq \frac{n-2}{2}\} \bigcup \{i \mid x_i \in S_{22}\} \bigcup \{i \mid x_i \in S_{1k}\} \quad , \qquad k=1 \text{ or } 2 \quad .$$

This is illustrated in Figure 4.2.7. Darkened nodes are in the elimination sequence. Nodes which are shaded have labels less than $\eta_{21}+j$. The following results are obtained using Lemma 3.3.4. If $x \in S_{21}$ and its labelling is $\eta_{21}+i$, $j \leq i \leq \frac{n-2}{2}$, then

$$\left| Reach_{\hat{G}}(x) \right| + 1 = n + 1 - i + j \quad .$$

If $x \in S_{22}$ and its labelling is $\eta_{22}+i$, $1 \leq i \leq \frac{n-2}{2}$, then

$$\left| Reach_{\hat{G}}(x) \right| + 1 = \frac{3}{2}n - i \quad .$$

If $x \in S_{1k}$ and its labelling is $\eta_{1k}+i$, $1 \leq i \leq n$, then

$$\left| Reach_{\hat{G}}(x) \right| + 1 = n + 1 - i \quad .$$

Thus the cost of eliminating this row is

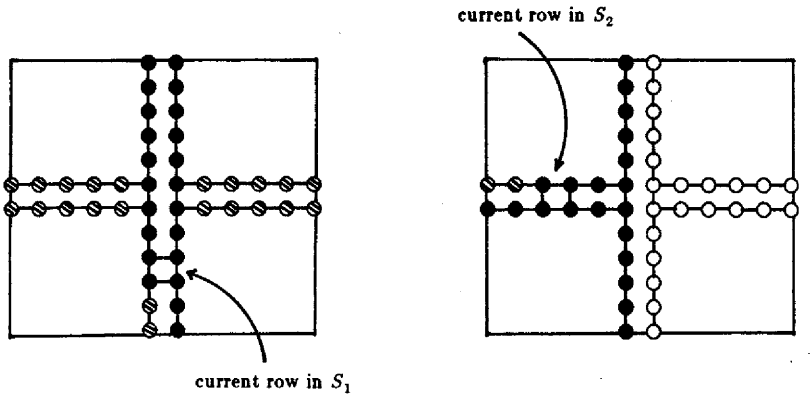


Figure 4.2.7 Darkened nodes are in the elimination sequence of a row.
Shaded nodes have labels less than ξ_1^k and are not in the elimination sequence.

$$\theta_{2j} = \sum_{i=j}^{\frac{1}{2}(n-2)} (n+1-i+j) + \sum_{i=1}^{\frac{1}{2}(n-2)} (\frac{3}{2}n-i) + \sum_{i=1}^{n} (n+1-i)$$

$$= \frac{3}{2}n^2 - \frac{1}{2}nj - \frac{3}{2}j - \frac{1}{2}j^2 \ .$$

Since there are $\frac{1}{2}(n-2)$ small squares in S_2, the total cost of eliminating the rows of S_2 is bounded by

$$\theta_2 = 4 \sum_{j=1}^{\frac{1}{2}(n-2)} \theta_{2j} = 4 \sum_{j=1}^{\frac{1}{2}(n-2)} (\frac{3}{2}n^2 - \frac{1}{2}nj - \frac{3}{2}j - \frac{1}{2}j^2) = \frac{8}{3}n^3 - 6n^2 + \frac{4}{3}n \ .$$

The factor 4 comes from the fact that there are four rows in each small square.

(7) Consider any row in S_1. Assume its elimination sequence is maximal. Let its leading subscript be $\eta_{11}+j$, $1 \leq j \leq n-1$. Using Lemma 3.3.5, the elimination sequence is

$$\{\eta_{11}+i \mid j \le i \le n\} \bigcup \{\eta_{12}+i \mid 1 \le i \le n\} \ .$$

This is illustrated in Figure 4.2.7. Nodes which are in the elimination sequence are darkened, and nodes whose labellings are less than $\eta_{11}+j$ are shaded. The following results are obtained using Lemma 3.3.4. If $x \in S_{11}$ and its labelling is $\eta_{11}+i$, $j \le i \le n$, then

$$\left| Reach_{\bar{G}}(x) \right| + 1 = n + 2 - i + j \ .$$

If $x \in S_{12}$ and its labelling is $\eta_{12}+i$, $1 \le i \le n$, then

$$\left| Reach_{\bar{G}}(x) \right| + 1 = n + 1 - i \ .$$

Thus the cost of eliminating this row is

$$\theta_{1j} = \sum_{i=j}^{n}(n + 2 - i + j) + \sum_{i=1}^{n}(n + 1 - i) = n^2 + 3n + 2 - \frac{3}{2}j - \frac{1}{2}j^2 \ .$$

Since there are $(n-1)$ small squares in S_1 and each small square has four rows, the total cost of eliminating the rows of S_1 is bounded by

$$\theta_1 = 4\sum_{j=1}^{n-1}\theta_{1j} = 4\sum_{j=1}^{n-1}(n^2 + 3n + 2 - \frac{3}{2}j - \frac{1}{2}j^2) = \frac{10}{3}n^3 + 6n^2 - \frac{4}{3}n - 8 \ .$$

Now we return to the derivation of $\theta(n,0)$. Recall that the cost of computing $R = R_0$ using rotations is given by

$$\theta(n,0) = \sum_k \alpha^k \ .$$

Using the previous observations, this can be written as

$$\begin{aligned} \theta(n,0) = \sum_k \alpha^k &= 4\theta(\frac{n-2}{2},2) + 2\theta_2 + \theta_1 \\ &= 4\theta(\frac{n-2}{2},2) + 2(\frac{8}{3}n^3 - 6n^2 + \frac{4}{3}n) + (\frac{10}{3}n^3 + 6n^2 - \frac{4}{3}n - 8) \\ &= 4\theta(\frac{n-2}{2},2) + \frac{26}{3}n^3 - 6n^2 + \frac{4}{3}n - 8 \ . \end{aligned} \qquad (4.2.15)$$

Using the same technique and arguments, the recurrence equations for $\theta(n,i)$, $i=2,3,4$, can also

be derived.

$$\theta(n,2) = \theta(\frac{n-2}{2},2) + 2\theta(\frac{n-2}{2},3) + \theta(\frac{n-2}{2},4) + \frac{413}{12}n^3 + 20n^2 + \frac{25}{3}n - 8 \ , \tag{4.2.16}$$

$$\theta(n,3) = 2\theta(\frac{n-2}{2},3) + 2\theta(\frac{n-2}{2},4) + \frac{167}{3}n^3 + 61n^2 + \frac{40}{3}n - 16 \ , \tag{4.2.17}$$

$$\theta(n,4) = 4\theta(\frac{n-2}{2},4) + \frac{245}{3}n^3 + 191n^2 + \frac{460}{3}n + 16 \ . \tag{4.2.18}$$

Recall that $n=2^m-2$. That is, $\frac{n-2}{2}=2^{m-1}-2$. Thus equations (4.2.15)-(4.2.18) can be rewritten in terms of m.

$$\theta(m,0) = 4\theta(m-1,2) + \frac{26}{3}2^{3m} - 58(2^{2m}) + \frac{388}{3}2^m - 104 \ , \tag{4.2.19}$$

$$\theta(m,2) = \theta(m-1,2) + 2\theta(m-1,3) + \theta(m-1,4) + \frac{413}{12}2^{3m} - \frac{373}{2}2^{2m} + \frac{1024}{3}2^m - 220 \ , \tag{4.2.20}$$

$$\theta(m,3) = 2\theta(m-1,3) + 2\theta(m-1,4) + \frac{167}{3}2^{3m} - 273(2^{2m}) + \frac{1312}{3}2^m - 244 \ , \tag{4.2.21}$$

$$\theta(m,4) = 4\theta(m-1,4) + \frac{245}{3}2^{3m} - 299(2^{2m}) + \frac{1108}{3}2^m - 180 \ . \tag{4.2.22}$$

The next lemma is useful when we determine a closed form for $\theta(m,0)$. It can be proved by induction.

Lemma 4.2.2

(a) If $f(m) = f(m-1) + \alpha 2^{3m} + O(m2^{2m})$, then $f(m) = \frac{8}{7}\alpha 2^{3m} + O(m2^{2m})$.

(b) If $g(m) = 2g(m-1) + \alpha 2^{3m} + O(m2^{2m})$, then $g(m) = \frac{4}{3}\alpha 2^{3m} + O(m2^{2m})$.

(c) If $h(m) = 4h(m-1) + \alpha 2^{3m} + O(2^{2m})$, then $h(m) = 2\alpha 2^{3m} + O(m2^{2m})$.

□

Now applying Lemma 4.2.2 (c) to (4.2.22), we have

$$\theta(m,4) = 4\theta(m-1,4) + \frac{245}{3}2^{3m} + O(2^{2m}) = \frac{490}{3}2^{3m} + O(m2^{2m}) \quad . \tag{4.2.23}$$

Applying Lemma 4.2.2 (b) and (4.2.23) to (4.2.21), we have

$$\theta(m,3) = 2\theta(m-1,3) + \frac{193}{2}2^{3m} + O(m2^{2m}) + \frac{386}{3}2^{3m} + O(m2^{2m}) \quad . \tag{4.2.24}$$

Using Lemma 4.2.2 (a), (4.2.23) and (4.2.24), (4.2.20) becomes

$$\theta(m,2) = \theta(m-1,2) + 87(2^{3m}) + O(m2^{2m}) = \frac{696}{7}2^{3m} + O(m2^{2m}) \quad . \tag{4.2.25}$$

Finally applying Lemma 4.2.2 (c) and (4.2.25) to (4.2.19), $\theta(m,0)$ is given by

$$\theta(m,0) = 4\theta(m-1,2) + \frac{26}{3}2^{3m} + O(m2^{2m}) = \frac{1226}{21}2^{3m} + O(m2^{2m}) \quad . \tag{4.2.26}$$

Replacing m and 2^m by $\log_2(n+2)$ and $(n+2)$ respectively and expanding $\log_2(n+2)$, we have

$$\theta(n,0) = \frac{1226}{21}n^3 + O(n^2\log_2 n) \quad . \tag{4.2.27}$$

Thus the cost of computing the upper triangular matrix R is $O(n^3)$.

The bound obtained in (4.2.27) may be too pessimistic because we are assuming that the elimination sequences of *all* the four rows associated with a small square are *maximal*. As we have shown in Theorem 3.4.3, this is not the case if the rows associated with a small square are eliminated *together*. Some of the elimination sequences are non-maximal, but the fourth one must be maximal. This discussion means that $\theta(n,0)$ must be *less than* $\frac{1226}{21}n^3 + O(n^2\log_2 n)$, *but* it must be *at least* $\frac{613}{42}n^3 + O(n^2\log_2 n)$. In any case, the order of magnitude remains $O(n^3)$.

Finally, as in the case of $\pi(n,0)$, the bounds in (4.2.15)-(4.2.18) are obtained by assuming a particular width-2 nested dissection ordering of the grid points. It is possible to derive a worst

case behavior without imposing such a restriction. For example, the elimination sequence of any row in S_2 is, in the worst case, given by

$$\Xi = \{i \mid x_i \in S_2\} \bigcup \{i \mid x_i \in S_{1k}\} \ , \quad k=1 \text{ or } 2 \ ,$$

and $|\Xi| = 2n-2$. Thus using Corollary 3.3.10, the cost of eliminating this row is at most

$$\theta_{2j} = \sum_{i=1}^{2n-2} i = \frac{1}{2}(2n-1)(2n-2) \ .$$

Note that there are $2(n-2)$ rows in S_2. Hence the total cost of eliminating the rows in S_2 is

$$\theta_2 = \sum_{j=1}^{2(n-2)} \theta_{2j} = \sum_{j=1}^{2(n-2)} \frac{1}{2}(2n-1)(2n-2) = 4n^3 - 14n^2 + 14n - 4 \ .$$

Similarly the elimination sequence of any row in S_1 is, in the worst case, given by

$$\Xi = \{i \mid x_i \in S_1\} \ ,$$

and $|\Xi| = 2n$. Thus the cost of eliminating this row is at most

$$\theta_{1j} = \sum_{i=1}^{2n} i = n(2n+1) \ .$$

Since there are $4(n-1)$ rows in S_1, the total cost of eliminating the rows in S_1 is

$$\theta_1 = \sum_{j=1}^{4(n-1)} \theta_{1j} = \sum_{j=1}^{4(n-1)} n(2n+1) = 8n^3 - 4n^2 - 4n \ .$$

Hence, in the worst case, $\theta(n,0)$ is given by

$$\begin{aligned} \theta(n,0) &= 4\theta(\frac{n-2}{2},2) + 2\theta_2 + \theta_1 \\ &= 4\theta(\frac{n-2}{2},2) + 16n^3 - 32n^2 + 24n - 8 \ . \end{aligned}$$

Using the same approach, it is possible to derive the worst case situations for $\theta(n,i)$, $i=2,3,4$.

$$\begin{aligned} \theta(n,2) = \theta(\frac{n-2}{2},2) + 2\theta(\frac{n-2}{2},3) + \theta(\frac{n-2}{2},4) + \frac{101}{2}n^3 \\ + \frac{1}{2}n^2 + 21n - 4 \ , \end{aligned}$$

$$\theta(n,3) = 2\theta(\frac{n-2}{2},3) + 2\theta(\frac{n-2}{2},4) + \frac{149}{2}n^3 + 43n^2 + 12n \ ,$$

$$\theta(n,4) = 4\theta(\frac{n-2}{2},4) + 104n^3 + 188n^2 + 148n + 40 \ .$$

Then one can show that, in the worst case,

$$\theta(n,0) = \frac{1760}{21}n^3 + O(n^2\log_2 n) \ . \tag{4.2.28}$$

As in the case of $\pi(n,0)$, the leading terms in both (4.2.27) and (4.2.28) are $O(n^3)$, but the coefficient in (4.2.28) is larger than that in (4.2.27). However it is obvious that in this case the worst case situation is not attainable.

We summarize our derivations in the following theorem.

Theorem 4.2.3

Let A be the rectangular matrix associated with an n by n grid and let R be the n^2 by n^2 upper triangular matrix obtained by reducing the rows of A using rotations. If the labelling of the grid points is a width-2 nested dissection labelling and if the row ordering induced by this width-2 nested dissection is used, then the number of nonzeros in R is $O(n^2\log_2 n)$ and the cost of computing R is $O(n^3)$.

□

To conclude this section, we present some numerical experiments on n by n grids. The objectives are to provide some specific instances of the bounds just obtained and to demonstrate the effectiveness of the row ordering induced by width-2 nested dissection. Some of the subroutines used in the experiments were taken from SPARSPAK-B. This package, which is an extension of SPARSPAK [30], is designed for solving large sparse systems of linear equations using rotation matrices and is still under development.

The column ordering used in the experiments was the one induced by width-2 nested dissection. The row orderings were the following.

(a) Row ordering A -- This is the row ordering induced by width-2 nested dissection. The rows are arranged so that the leading subscripts are in ascending order. (That is, the rows associated with the separators are eliminated *last*.)

(b) Row ordering B -- The rows are arranged so that the leading subscripts are in descending order. (That is, the rows associated with the separators are eliminated *first*.)

(c) Row ordering C -- This is the so-called natural row ordering. The rows are collected from the small squares in the grid row by row.

The following notation is used in Tables 4.2.1 and 4.2.2.

N -- number of columns ($N=n^2$).

M -- number of rows ($M=4(n-1)^2$).

NZ -- number of nonzeros ($NZ=16(n-1)^2$).

S -- number of storage locations required for the upper triangular matrix R.

τ_A -- transformation time for row ordering A (in seconds).

τ_B -- transformation time for row ordering B (in seconds).

τ_C -- transformation time for row ordering C (in seconds).

The results in Table 4.2.1 confirm that if the column and row orderings induced by width-2 nested dissection are used, the storage requirement and transformation time are $O(n^2\log_2 n)$ and $O(n^3)$ respectively. Note that S includes the storage required to store the nonzeros of R *and* the structure of R (that is, pointers and column indices).

The results in Table 4.2.2 show that the row ordering induced by width-2 nested dissection (that is, row ordering A) is indeed better than the other two row orderings used.

More numerical results will be presented later in this chapter.

n	N	M	NZ	S	$\frac{S}{n^2\log_2 n}$	τ_A	$\frac{\tau_A}{n^3}$
10	100	324	1296	2223	6.69	1.230	0.00123
12	144	484	1936	3419	6.62	2.297	0.00133
14	196	676	2704	5058	6.78	3.823	0.00139
16	256	900	3600	7139	7.02	6.153	0.00150
18	324	1156	4624	9805	7.26	8.906	0.00153
20	400	1444	5776	12679	7.33	12.416	0.00155
22	484	1764	7056	16076	7.45	16.596	0.00156
24	576	2116	8464	19509	7.39	21.595	0.00156
26	676	2500	10000	23771	7.48	28.725	0.00163
28	784	2916	11664	28650	7.60	36.364	0.00166
30	900	3364	13456	34093	7.72	46.224	0.00171

Table 4.2.1 Storage requirement and execution time for width-2 nested dissection orderings.

n	N	M	NZ	τ_A	τ_B	τ_C
10	100	324	1296	1.230	1.763	1.407
12	144	484	1936	2.297	3.736	2.846
14	196	676	2704	3.823	6.923	5.193
16	256	900	3600	6.153	11.749	8.449
18	324	1156	4624	8.906	18.815	13.429
20	400	1444	5776	12.416	27.422	20.145
22	484	1764	7056	16.596	40.877	29.288
24	576	2116	8464	21.595	56.699	40.307
26	676	2500	10000	28.725	79.248	55.037
28	784	2916	11664	36.364	106.056	72.135
30	900	3364	13456	46.224	141.788	97.634

Table 4.2.2 Execution times for three different row orderings.

4.3. Generalized width-2 nested dissection

Suppose $G=(X,E)$ is a finite element graph satisfying the conditions of Theorem 3.5.4 and assume $|X|=n$. Then we have shown that the node set X can be partitioned into A, B and C such that C is a width-2 separator having $O(\sqrt{n})$ nodes. The graphs $G(A)$ and $G(B)$ are disconnected, and neither A nor B contains more than $\frac{2}{3}n$ nodes. Because of our definition of finite element graphs, $G(A \bigcup Adj(A))$ and $G(B \bigcup Adj(B))$ also satisfy the conditions of Theorem 3.5.4. Hence, one can partition $A \bigcup Adj(A)$ and $B \bigcup Adj(B)$ similarly. In fact, the process can be repeated recursively and it produces a width-2 nested dissection partitioning.

Suppose the nodes are labelled in such a way that the labelling is a width-2 nested dissection ordering with respect to the width-2 nested dissection partitioning. Consider the finite element problem defined on G. Let R be the n by n upper triangular matrix obtained in Algorithm 2.3.1. The rows of the problem are assumed to be arranged so that the leading subscripts are in ascending order. Then it is possible to show that the number of nonzeros in R and the cost of computing R are respectively $O(n^{\frac{3}{2}})$ and $O(n\log n)$, as long as there is a bounded number of equations defined on each element. The proofs are omitted since they are similar to those in Section 4.2 and [40, 56]. Note that the results are consistent with those of the model problem.

4.4. Automatic width-2 nested dissection

The success of width-2 nested dissection hinges on the ability to find width-2 separators in the subgraphs of the graph of A^TA. For problems arising from n by n grids, we have seen that it is very easy to find small width-2 separators. For problems arising from two-dimensional finite element graphs whose skeletons are planar graphs, Lipton and Tarjan have proposed an algorithm for finding small width-1 separators [57]. This can be used as a basis for finding width-2 separators (see Section 3.5). Unfortunately their algorithm is very complicated. As far as we know, no actual implementation exists yet. For more general problems, there is no known

algorithm that will guarantee to find small width-1 separators. Thus in order to implement the width-2 nested dissection algorithm for labelling general graphs, we need some heuristic methods for finding width-1 and width-2 separators. In this section we describe one such heuristic method. Experience has shown that for two-dimensional finite element problems, the separators are usually small. The method has been used extensively by George and Liu in the automatic generation of symmetric orderings for large sparse symmetric positive definite matrices [36]. Examples include the width-1 nested dissection [28], the one-way dissection [24], and the refined quotient tree orderings [29]. It is based on a so-called rooted level structure [2].

Let A be an m by n matrix with $m \geq n$. Let $G=(X,E)$ be the graph of A^TA. A *level structure rooted at a node* r, denoted by

$$L(r) = \{L_0(r), L_1(r), \cdots, L_{l_r}(r)\} \ ,$$

is a partitioning of the node set X that satisfies the following conditions.

(a) $\bigcup_{k=0}^{l_r} L_k(r) = X$.

(b) For $i \neq j$, $L_i(r) \bigcap L_j(r) = \emptyset$.

(c) $L_0(r)=\{r\}$, $L_1(r)=Adj(L_0(r))$, and $L_k(r)=Adj(L_{k-1}(r))-L_{k-2}(r)$, for $2 \leq k \leq l_r$.

That is, for $1 \leq k \leq l_r-1$, $Adj(L_k(r)) \subseteq L_{k-1}(r) \bigcup L_{k+1}(r)$. Thus for $0<k<l_r$, $L_k(r)$ is a *width-1 separator* (even though it may not be minimal). A minimal width-1 separator S can be obtained from $L_k(r)$ simply by finding those nodes in $L_k(r)$ that have neighbors in *both* $L_{k-1}(r)$ and $L_{k+1}(r)$. A minimal width-2 separator is then given by $S \bigcup (Adj(S) \bigcap L_{k-1}(r))$ or $S \bigcup (Adj(S) \bigcap L_{k+1}(r))$.

How do we choose the node r? A simple solution to this problem is the following. Let $w_r = \frac{n}{l_r}$, where $n = |X|$. That is, w_r is the average number of nodes in each level. Now we may regard the level structure rooted at r as a w_r by l_r grid. From the discussion in Section 4.2, we would choose as a width-1 separator the "grid-line" $L_k(r)$ where $k \approx \frac{l_r}{2}$. Hopefully $L_k(r)$ will be

small if l_r could be made large.

How can we make l_r large? Define the *eccentricity* of a node x [7] to be the quantity

$$e(x) = \max\{d(x,y) \mid y \in X\} \ .$$

The diameter of G is then defined as

$$\delta = \max\{e(x) \mid x \in X\} \ ;$$

that is, the diameter is the largest distance between any two nodes in the graph. Thus one can choose r so that its eccentricity is the same as the diameter of G. Such a node is called a *peripheral node*. However the best known algorithm for finding peripheral node in a given graph $G=(X,E)$ has a time complexity bound of $O(|X||E|)$ [70]. This is too expensive for sparse matrix applications.

Instead of using a peripheral node, we use a so-called *pseudo-peripheral node* which is a node that has large eccentricity. Many efficient heuristic algorithms are available for finding pseudo-peripheral nodes [32, 39]. The one we use here is due to George and Liu [32, 59].

We describe below an algorithm that automatically determines a width-2 separator in the graph G. For simplicity we assume G is connected.

Algorithm 4.4.1

(1) Find a pseudo-peripheral node r in G using the algorithm from [32].

(2) Generate a level structure rooted at r, $L(r)=\{L_0(r),L_1(r),\cdots,L_{l_r}(r)\}$.

(3) If $l_r \geq 3$, then choose a minimal width-2 separator from levels $\left\lfloor \frac{l_r}{2} \right\rfloor$ and $\left\lfloor \frac{l_r}{2} \right\rfloor + 1$. Otherwise, the whole set X is a width-2 separator.

Note that different ways of determining the width-2 separator in step 3 will generate different width-2 nested dissection orderings. Experience has shown that these orderings differ only slightly in most cases in terms of nonzero count in R and time for computing R.

Problem number	Number of rows	Number of columns	Number of nonzeros	Remarks
1	219	85	438	Holland survey data.
2	958	292	1916	U.K. survey data.
3	331	104	662	Scotland survey data.
4	608	188	1216	England survey data.
5	313	176	1557	Sudan survey data.
6	1033	320	4732	A well conditioned least squares problem in the analysis of gravity-meter observations (provided by M.A. Saunders).
7	1033	320	4719	An ill-conditioned problem whose structure is similar to that of problem 2.
8	1850	712	8755	Similar to problem 2, but larger in size.
9	1850	712	8636	Problem similar in structure to problem 4, but ill-conditioned.
10	784	225	3136	Artificial - 15 by 15 grid problem.
11	1444	400	5776	Artificial - 20 by 20 grid problem.
12	1512	402	7152	Artificial - 3 by 3 network, μ=2.
13	1488	784	7040	Artificial - 4 by 4 network, μ=1.
14	900	269	4208	A geodetic network problem provided by the U.S. National Geodetic Survey (ill-conditioned).

Table 4.4.1 Characteristics of the test problems.

We now present some numerical experiments to demonstrate the effectiveness of automatic width-2 nested dissection orderings. The test set consists of fourteen problems, some of which are real problems while the others were generated artificially. They are large sparse overdetermined systems of linear equations. Problems 5 to 14 were used in [26]. There are two types of artificially generated problems.

(a) The first type of artificially generated problem involves a network which is typical of those arising in geodetic adjustment applications [54]. Problems 12 and 13 belong to this class. Each network may be regarded as being composed of q^2 "junction boxes", connected to their neighbors by chains of length l. An example is shown in Figure 4.4.1 with q=3 and

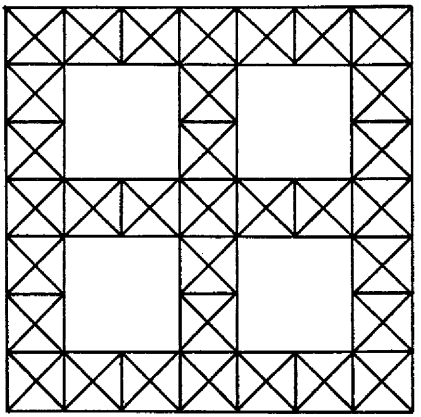


Figure 4.4.1 A 3 by 3 network with l=2.

l=2. There are two variables associated with each vertex in the network. There are μ observations involving four variables associated with each pair of vertices joining by an edge, and there are μ additional observations involving six variables associated with each triangle in the network.

(b) The model grid problem is the second type of artificially generated problem. Problems 10 and 11 are examples of it.

Table 4.4.1 contains some characteristics of each of the fourteen problems. For the artificially generated problems, the numerical values in the observation matrices and the right-hand sides were generated using a uniform random number generator.

problem	N	M	NZ	S	τ_A	τ_B	τ_C	τ_D
1	85	219	438	1690	0.700	0.960	0.787	0.780
2	292	958	1916	7706	7.136	10.803	7.516	7.540
3	104	331	662	2210	1.367	2.057	1.337	1.350
4	188	608	1216	4885	3.803	6.293	4.476	4.326
5	176	313	1557	4352	1.443	1.873	1.590	1.923
6	320	1033	4732	13631	38.696	25.043	48.823	26.693
7	320	1033	4719	13684	38.533	28.240	49.363	26.876
8	712	1850	8755	48857	328.613	235.047	350.186	308.910
9	712	1850	8636	48954	330.266	492.564	378.775	426.165
10	225	784	3136	6135	5.037	10.410	8.280	8.557
11	400	1444	5776	12389	12.223	33.556	28.876	28.126
12	402	1512	7152	10732	11.340	16.286	15.403	27.843
13	784	1488	7040	24550	16.183	25.103	19.417	42.679
14	269	900	4208	11545	20.210	23.890	24.886	20.110

Table 4.4.2 Storage requirement in width-2 nested dissection column ordering, and execution times for four different row orderings.

The column ordering used in the experiments was the width-2 nested dissection ordering which was generated using the automatic scheme described in this section. Four row orderings were investigated.

(a) Row ordering A -- The rows are arranged so that the leading subscripts are in ascending order. This is the row ordering induced by width-2 nested dissection.

(b) Row ordering B -- The rows are arranged so that the leading subscripts are in descending order.

(c) Row ordering C -- This is the initial row ordering provided.

(d) Row ordering D -- The rows are arranged so that the numbers of nonzeros in the rows are in ascending order.

We include row ordering D because some believe that the upper triangular matrices in the intermediate stages will be less dense if rows with smaller number of nonzeros are eliminated first. This might imply that the overall cost of computing the upper trapezoidal form would then be smaller. However this is not necessarily true, since the distribution of the nonzeros is important as well. The intermediate upper triangular matrices could be dense, and the elimination of some of the rows may be very expensive. The objective here is to illustrate this possibility.

Table 4.4.2 contains the storage required for the upper triangular matrices and the transformation times (in seconds) for the various row orderings. For convenience the following notation will be used in Table 4.4.2.

M -- number of rows.

N -- number of columns.

NZ -- number of nonzeros.

S -- number of storage locations required for the upper triangular matrix.

τ_A -- transformation time for row ordering A (in seconds).

τ_B – transformation time for row ordering B (in seconds).

τ_C – transformation time for row ordering C (in seconds).

τ_D – transformation time for row ordering D (in seconds).

A few remarks on the numerical results must be made.

(1) The numerical results confirm the fact that row ordering D (arranging the nonzero counts in ascending order) may not be a good one. For some problems, the transformation times are even larger than those when the initial row ordering (row ordering C) is used.

(2) Row ordering A (the one induced by width-2 nested dissection) is the best row ordering for nine of the fourteen problems.

problem	N	size of separator
1	85	19
2	292	22
3	104	16
4	188	30
5	176	11
6	320	277
7	320	277
8	712	490
9	712	490
10	225	32
11	400	40
12	402	40
13	784	56
14	269	42

Table 4.4.3 Size of width-2 separators in the first level of dissection.

(3) Both storage requirements and transformation times for problems 8 and 9 are large. Furthermore row ordering A is not the best row ordering for problems 6, 7 and 8. One possible reason is as follows. The way in which the width-2 separators are obtained is heuristic. There is no guarantee that the width-2 separators are small. If the width-2 separators are large, then our discussion in Section 1 of this chapter implies that the storage requirement could be large. Furthermore, the number of rows associated with a large width-2 separator could also be large and the cost of eliminating these rows could be high. We have listed in Table 4.4.3 the size of the width-2 separator for each problem in the first level of dissection. Notice that for problems 6, 7, 8 and 9, each separator contains more than half of the nodes in the graph. This explains why the storage requirements and transformation times are large for those four problems. For the other problems, the width-2 separators are relatively small; each contains less than $\frac{1}{4}$ of the nodes in the graph. This illustrates the main disadvantage of automatic width-2 nested dissection algorithm: the induced row and column orderings are good (in terms of storage requirement and execution time) only when one can find *small* width-2 separators in the subgraphs of G. However only a few classes of graphs, which include planar and some two-dimensional finite element graphs, can be guaranteed to have small width-2 separators.

(4) Recall from Chapter 3 that the number of nonzeros in the upper triangular matrix predicted by our model may be pessimistic. That is, it may be much larger than the actual number of nonzeros produced in the numerical phase. This would be particularly true for general sparse problems. In order to obtain some idea on how good our model is, we have counted the actual number of nonzeros produced in the transformation for each problem and compared it with the number of nonzeros predicted by the symbolic factorization procedure. Surprisingly, for our set of test problems, the number of nonzeros produced in the transformation process was *exactly* equal to that predicted by the model, except for problems 6 and 8. Even for problems 6 and 8, the number of nonzeros produced was very

close to the predicted number (99.9%).

CHAPTER 5

WIDTH-ONE NESTED DISSECTION ORDERINGS

In Chapter 4 we showed that, for an m by n matrix A, we identify a set of rows if we can find a small minimal width-2 separator in the graph of A^TA. The important thing about these rows is that they are relatively cheap to eliminate if they are eliminated last. Moreover the amount of fill-in is small. By applying this idea recursively we obtain a width-2 nested dissection partitioning and a width-2 nested dissection ordering. As we have shown, this ordering induces good column and row orderings in the orthogonal decomposition of A using rotations. However experience shows that the amount of fill-in in the upper triangular matrix R (that is, the Cholesky factor of A^TA) is often larger than that obtained when the columns of A (or the columns and rows of A^TA) are labelled by other ordering algorithms, such as the minimum degree algorithm or even the width-1 nested dissection algorithm. This is particularly true for general sparse problems. In this chapter we consider the width-1 nested dissection algorithm. We show that there is a good row ordering induced by a width-1 nested dissection column ordering. We show that, for the model problem, the number of nonzeros in R and the cost of computing it are respectively $O(n^2\log_2 n)$ and $O(n^3)$ if the orderings induced by width-1 nested dissection are used. Empirical results that suggest that a minimum degree ordering may be a width-1 nested dissection ordering are provided.

5.1. Width-1 nested dissection

Width-1 nested dissection (or simply nested dissection) was first proposed by George as a technique for ordering the columns and rows of a sparse symmetric positive definite matrix that arises in the application of finite difference and finite element methods on square grids [23]. It is known that the number of nonzeros in the Cholesky factor of the reordered matrix and the number of multiplicative operations in the computation are both optimal (in the order of

magnitude sense) [52]. In [28] an automatic scheme was proposed for generating a nested dissection ordering for irregular finite element meshes. Related work and analyses can be found in [8, 16, 56, 66]. Width-1 nested dissection is almost identical to width-2 nested dissection; the only difference is that, in width-1 nested dissection, the separators are width-1 separators.

Let A be an m by n matrix, with $m \geq n$. Denote the n by n upper triangular matrix obtained in the orthogonal decomposition of A by R. Recall from Chapter 2 that the structure of R is assumed to be identical to that of the Cholesky factor of the symmetric positive definite matrix A^TA. Thus we can use a width-1 nested dissection algorithm to relabel the columns of A (or the columns and rows of A^TA). Assume the algorithm produces a width-1 nested dissection column ordering (we will see later than this is possible if the graph of A^TA satisfies certain conditions). In this section, we show that a width-1 nested dissection column ordering also induces a good row ordering and that there is a simple characterization of the induced row ordering.

We first describe the width-1 dissection technique. Let $G=(X,E)$ be the graph of the n by n matrix A^TA. We assume G is connected. Let S be a width-1 separator of G whose removal disconnects the graph into two or more components. Assume there are two components and denote their node sets by C_1 and C_2 respectively. In width-1 dissection, the nodes of C_1 and C_2 are labelled *before* those of S. Let x_k denote the node having labelling k. Then $p<q$ if $x_p \in C_i$, $i=1,2$, and $x_q \in S$. The node labelling strategy, together with Lemma 3.3.1, implies the following. Suppose $x_i \in C_1$, $x_j \in C_2$ and $i<j$. Then $R_{ij}=0$ since there cannot be a path connecting x_i and x_j that involves nodes whose labellings are less than i. Hence the dissection technique based on width-1 separators guarantees that the amount of fill-in in R is low. The technique can be used to label the nodes of C_1 and C_2 recursively, yielding a width-1 nested dissection ordering.

For completeness we now define formally *width-1 nested dissection partitionings* which are similar to width-2 nested dissection partitionings. Let $G=(X,E)$ be the unlabelled graph of

A^TA. Let $Y^0=X$, and for $m=0,1,2,\cdots,h$ until $Y^{h+1}=\emptyset$, do the following:

(1) Determine the connected components of Y^m and label them $Y_1^m, Y_2^m, \cdots, Y_{r_m}^m$.

(2) For $j=1,2,\cdots,r_m$, choose $\overline{S}_j^m$ such that $\overline{S}_j^m$ is a width-1 separator of $G(Y_j^m)$. If $\overline{S}_j^m \neq \emptyset$, set $S_j^m=\overline{S}_j^m$. Otherwise, set $S_j^m=Y_j^m$.

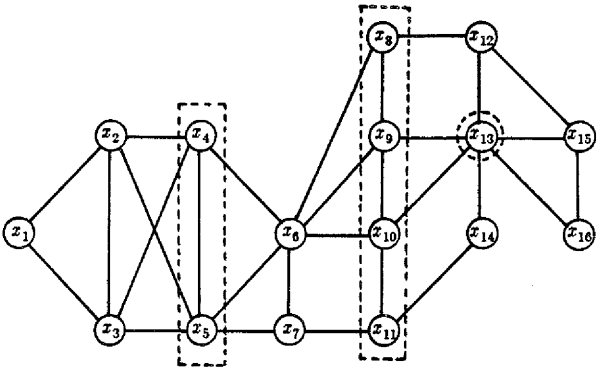
(3) Define $S^m=\bigcup_{j=1}^{r_m} S_j^m$ and $Y^{m+1}=Y^m-S^m$.

The partitioning $\Phi=\{S_j^m \subseteq X, 1\leq j\leq r_m, 0\leq m\leq h\}$ is called a width-1 nested dissection partitioning.

An ordering of X is said to be a *width-1 nested dissection ordering* with respect to $\Phi=\{S_j^m\}$ if for $x\in S_j^m$ and $y\in Y_j^m-S_j^m$, x is labelled *after* y. An example of a width-1 nested dissection ordering with respect to the partitioning of Figure 5.1.1 is shown in Figure 5.1.2. It is possible to obtain a bound on the number of off-diagonal nonzeros in R by replacing width-2 separators by width-1 separators in Theorem 4.1.7. Note that it is desirable to choose small separators, since the bound depends on the size of the separators.

Throughout our discussion, the graph (and its subgraphs) are assumed to have separators. This is not true in general. Planar and certain two-dimensional finite element graphs can be guaranteed to have small separators (compared to the size of the graph) [57]. General graphs may not necessarily have separators; even if they do, the separators need not be small.

We now return to the row ordering problem. Unfortunately, unlike width-2 nested dissection, a width-1 separator S or the section subgraph $G(S)$ *need not* identify a set of rows. This is illustrated by an example in Figure 5.1.3. Here $S=\{x_2,x_6\}$ is a width-1 separator, but there are not any rows of A such that their row graphs are in the subgraph $G(S)$. That is, $G(S)$ does not contain any row graph. Thus, apparently width-1 nested dissection does not immediately provide information on how the rows of A should be labelled.



$$Y_1^0 = X \qquad S_1^0 = \{x_8, x_9, x_{10}, x_{11}\}$$

$$Y_1^1 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \qquad S_1^1 = \{x_4, x_5\}$$

$$Y_2^1 = \{x_{12}, x_{13}, x_{14}, x_{15}, x_{16}\} \qquad S_2^1 = \{x_{13}\}$$

$$Y_1^2 = S_1^2 = \{x_1, x_2, x_3\}$$

$$Y_2^2 = S_2^2 = \{x_6, x_7\}$$

$$Y_3^2 = S_3^2 = \{x_{14}\}$$

$$Y_4^2 = S_4^2 = \{x_{12}, x_{15}, x_{16}\}$$

Figure 5.1.1 A width-1 nested dissection partitioning $\{S_j^m\}$ of a graph of $A^T A$.

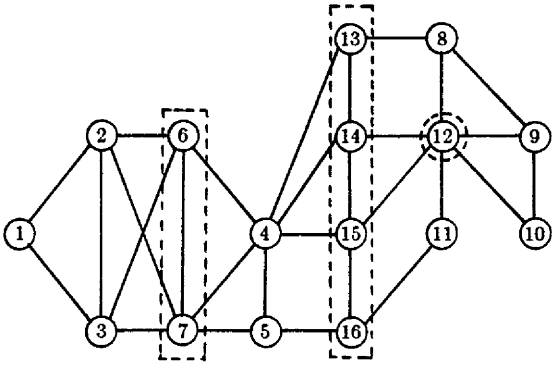


Figure 5.1.2 A width-1 nested dissection ordering on
the width-1 nested dissection partitioning of Figure 5.1.1.

Now recall that G is the union of m row graphs, $\phi^k=(\chi^k,\epsilon^k)$, each of which is a complete graph. Lemma 4.1.1 indicates that each χ^k is contained in either $C_1 \bigcup S$ or $C_2 \bigcup S$, where S is a separator. Thus, intuitively, the subgraph $G(S \bigcup Adj(S))$ should contain some of the row graphs ϕ^k. If this is the case, then a width-1 separator *does* provide us with a mechanism for studying the row ordering problem. Before we prove that $G(S \bigcup Adj(S))$ is the union of some row graphs, we consider the example in Figure 5.1.3. Here $S \bigcup Adj(S)=\{x_2,x_3,x_5,x_6,x_8\}$ and $G(S \bigcup Adj(S))$ contains the row graphs of rows 3, 4, 8 and 10.

Lemma 5.1.1

If S is a width-1 separator, then there is at least one row graph in the section graph $G(S \bigcup Adj(S))$.

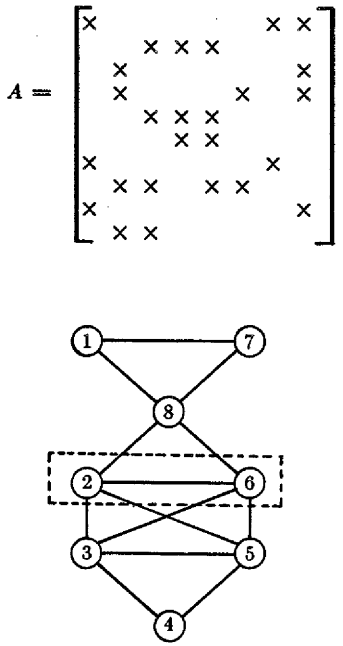


Figure 5.1.3 An example illustrating the fact that a width-1 separator may not identify any row.

Proof:

Let $x \in S$. Since $G = \bigcup_{i=1}^{m} \phi^i$, there must exist one row graph, say $\phi^k = (\chi^k, \epsilon^k)$, such that $x \in \chi^k$. Note that $\chi^k - \{x\} \subseteq Adj(x) \subseteq S \bigcup Adj(S)$ since χ^k is a clique in the graph G. Hence the result follows.

□

Lemma 5.1.2

Let S be a width-1 separator and $\phi^k=(\chi^k,\epsilon^k)$ be a row graph. If $\chi^k \bigcap S \neq \emptyset$, then either $\chi^k \subseteq S \bigcup (Adj(S) \bigcap C_1)$ or $\chi^k \subseteq S \bigcup (Adj(S) \bigcap C_2)$.

Proof:

This follows from the fact that χ^k is a clique and S is a width-1 separator.

□

Consequently there is no row graph $\phi^k=(\chi^k,\epsilon^k)$ such that χ^k contains nodes from C_1, C_2 and S.

Theorem 5.1.3

If S is a width-1 separator, then $G(S \bigcup Adj(S))$ is the union of one or more row graphs.

Proof:

By Lemma 5.1.1, there is at least one row graph in the section graph $G(S \bigcup Adj(S))$. Then the result follows from Lemma 5.1.2 and the observation that any node of S must belong to at least one χ^k.

□

The fact that $G(S \bigcup Adj(S))$ is the union of some row graphs provides us with a mechanism of finding a "good" row ordering. Consider the following strategy.

Let Z_3 be the set of row graphs in $G(S \bigcup Adj(S))$. For $i=1,2$, let Z_i be the set of row graphs remaining in $G(C_i)$. Thus $Z_1 \bigcup Z_2 \bigcup Z_3$ is the set of m row graphs. Moreover Z_1, Z_2 and Z_3 are disjoint. As in Section 4.1, we use $\chi(Z_i)$ to denote the union of the node sets of the row graphs in Z_i. Hence $\chi(Z_1) \bigcup \chi(Z_2) \bigcup \chi(Z_3)=X$. Note that $\chi(Z_1) \subseteq C_1$ and $\chi(Z_2) \subseteq C_2$. Thus $\chi(Z_1) \bigcap \chi(Z_2)=\emptyset$, but $\chi(Z_3) \bigcap \chi(Z_1) \neq \emptyset$ and $\chi(Z_3) \bigcap \chi(Z_2) \neq \emptyset$. In fact, for $i=1,2$, $\chi(Z_3) \bigcap \chi(Z_i)=C_i \bigcap Adj(S)$. We now propose eliminating the rows of Z_1 and Z_2 *before* those of Z_3. It is easy to understand why it is desirable to eliminate the rows of Z_1 and Z_2 first. The argument is the same as that used in Section 4.1.

Denote the component $G(C_i)$ by G_i, i=1,2. The first observation is that G_1 and G_2 are disjoint unless the row graphs of Z_3 are merged with G_1 and G_2. In other words, the elimination of any row of Z_i involves only the nodes of C_i, i=1,2. This is due to Theorem 3.3.8. Thus, by eliminating the rows of Z_3 last, the elimination sequence of any row of Z_1 or Z_2 is limited to a proper subgraph of G. Moreover the elimination sequences of the rows of Z_1 and the elimination sequences of the rows of Z_2 are independent, regardless of the order in which these rows are eliminated. Note that the nodes of C_1 and C_2 are also labelled recursively using the width-1 dissection technique. Thus the rows of Z_1 and Z_2 can be reordered using the same strategy recursively.

How about the elimination of the rows of Z_3? Suppose $\phi^k=(\chi^k,\epsilon^k)$ is a row graph of Z_3. Let Ξ^k be the elimination sequence of the row corresponding to ϕ^k and s the leading subscript of this row. From Lemma 2.1.4, s is therefore the first member of Ξ^k. Let x_k denote the the node of X having labelling k. Denote the set of nodes $\{x_i \in X \mid i \in \Xi^k\}$ by Δ^k, which will also be referred to as the elimination sequence of ϕ^k. There are two possibilities.

The first possibility is that $x_s \in S$. Then Lemma 3.3.5 tells us that the set Δ^k is just a subset of S since the labellings of the nodes of C_1 and C_2 are less than s. Hence the length of the elimination sequence will be small (if S is small). Note that this is possible only when χ^k is entirely contained in S, because of the node labelling strategy and because S is a width-1 separator.

The second possibility is that either $x_s \in C_1 \bigcap Adj(S)$ or $x_s \in C_2 \bigcap Adj(S)$. Note that s may be much smaller than the labelling of any node of S. Using Theorem 3.3.8, the set that contains Δ^k would be as large as $C_1 \bigcup C_2 \bigcup S$ and the elimination sequence would be long. This is undesirable. However, since the labellings of the nodes of S are the largest, any path joining $x_i \in C_1$ and $x_j \in C_2$ *must* contain a node $x_k \in S$ where $k>i$ and $k>j$. Thus, using Lemma 3.3.5, the set that contains Δ^k would only be as large as $C_1 \bigcup S$ or $C_2 \bigcup S$, depending on whether $x_s \in C_1$ or $x_s \in C_2$. Note that $|C_1 \bigcup S|$ or $|C_2 \bigcup S|$ would still be large compared to $|S|$.

Hence eliminating the rows of Z_3 last may be expensive.

Since the nodes of C_1 and C_2 are also labelled using the width-1 dissection technique, the elimination sequence Δ^k is only a subset of $C_i \bigcup S$ even if $x_s \in C_i \bigcap Adj(S)$. In order to have a better understanding of the elimination of the rows of Z_3, we partition the separators according to the level of dissection. A separator is a *level-k separator* if it is obtained in the *k–th* level of dissection. In the example given in Figure 5.1.2, $\{14,15,16,17\}$ is a level-1 separator, $\{7,8\}$ and $\{12,13\}$ are level-2 separators and $\{6\}$ is a level-3 separator.

Now suppose ϕ^k is in $G(S \bigcup Adj(S))$ and assume x_s is in a level-i separator, say S_i, $i \geq 1$. In the worst case the entire level-i separator will be included in Δ^k. The level-i separator is obtained from a connected component created in the $(i-1)$–*st* dissection. The component is surrounded, in part, by a level-$(i-1)$ separator, say S_{i-1}. (It is also surrounded by the level-1 separator $S_1 = S$ and some level-j separators, S_j, $1 < j < i$, which will eventually be considered.) Thus the level-$(i-1)$ separator S_{i-1} is part of Δ^k. Using the same argument repeatedly, one can see that the set Δ^k involves, in the worst case, *exactly* one level-j separator S_j, for $1 \leq j \leq i$. That is, Δ^k does not include the entire set $C_1 \bigcup S$ or $C_2 \bigcup S$. It is a proper subset of $C_1 \bigcup S$ or $C_2 \bigcup S$ even in the worst case. (Here we assume that the graphs $G(C_1)$ and $G(C_2)$, and the subsequent subgraphs have separators.) This is illustrated by an example in Figure 5.1.4 where the separators included in Δ^k are darkened. In other words, the length of the elimination sequence of any row of Z_3 is limited. Moreover the reachable set of any node on a level-j separator S_j contains only nodes of those separators immediately surrounding S_j (see Figure 5.1.4). Thus the cost of eliminating the rows of Z_3 could be low if the separators are small.

As in the case of width-2 nested dissection, since we are working with the graph of $A^T A$, identifying the sets S and $Adj(S)$ does not immediately provide us with the information for ordering the rows. Fortunately, as we are going to show, the induced row ordering can be obtained easily. Define the *last subscript* of a row to be the column index of the last nonzero in that row, or equivalently, the last subscript of a row graph is the labelling of the node having the

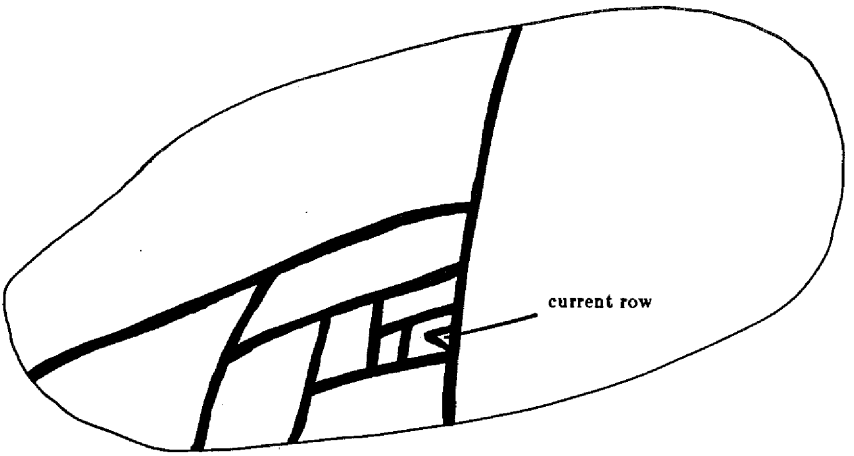


Figure 5.1.4 The nodes involved in the elimination of a row in $G(S \bigcup Adj(S))$ belong to separators which are darkened.

largest labelling. Suppose ϕ^p and ϕ^q are respectively row graphs of Z_3 and Z_i $(i=1,2)$. Because of the node labelling strategy, the node having the last subscript of ϕ^p must be in S, while that of ϕ^q is in C_i. That is, the last subscript of ϕ_p must be *greater than* the last subscript of ϕ^q. Hence the row ordering induced by width-1 nested dissection is obtained simply by arranging the rows of the (column-permuted) matrix A so that *the last subscripts are in ascending order.*

We now summarize our discussion below. Let A be an m by n matrix with $m \geq n$.

(1) Find a width-1 nested dissection column ordering for A. Denote the ordering by P.

(2) Permute the rows of AP so that the last subscripts are in ascending ordering.

Theorem 4.1.6 can be modified easily to provide an upper bound on the cost of computing the upper triangular matrix R. Let $\{S_j^m, 1 \le j \le r_m, 0 \le m \le h\}$ be a width-1 nested dissection partitioning. Note that S_j^m, $1 \le j \le r_m$, are level-$(m+1)$ separators. Let Z_j^m be the set of row graphs associated with $S_j^m \bigcup Adj(S_j^m)$. Our discussion above shows that the elimination sequence includes S_j^m and exactly one level-$(k+1)$ separator, say S_i^k, $m < k \le h$. Thus

$$\mu_j^m \le |S_j^m| + \sum_{k=m+1}^{h} \max_{1 \le i \le r_k} |S_i^k| ,$$

and the cost of computing R is given by

$$\sum_{m=0}^{h} \sum_{j=1}^{r_m} \frac{|Z_j^m| \mu_j^m (\mu_j^m + 1)}{2} .$$

5.2. Complexity of width-1 nested dissection for the model problem

Consider the model problem of Section 2.4. Let A be the rectangular matrix associated with an n by n grid and R be the upper triangular matrix obtained from A using row elimination. We now show that if the columns of A (or the grid points) and the rows of A are ordered by width-1 nested dissection, then the number of nonzeros in R and the cost of computing R are respectively $O(n^2 \log_2 n)$ and $O(n^3)$, which are the same as those in width-2 nested dissection.

As in Section 4.2, we will make use of bordered n by n grids. Let R_i denote the upper triangular matrix associated with an n by n grid bordered along i sides. Let $\pi(n,i)$ and $\theta(n,i)$ denote respectively the number of off-diagonal nonzeros in R_i and the cost of computing R_i. Thus $\pi(n,0)$ and $\theta(n,0)$ are the quantities we want to determine. For simplicity we will assume that $n = 2^m - 1$, for some integer $m > 1$.

We first review the width-1 dissection technique. In the first level of dissection, a vertical grid-line containing n grid points is chosen as the level-1 width-1 separator. It dissects the unbordered n by n grid into two n by $\frac{1}{2}(n-1)$ grids. Then, in the second level of dissection, a

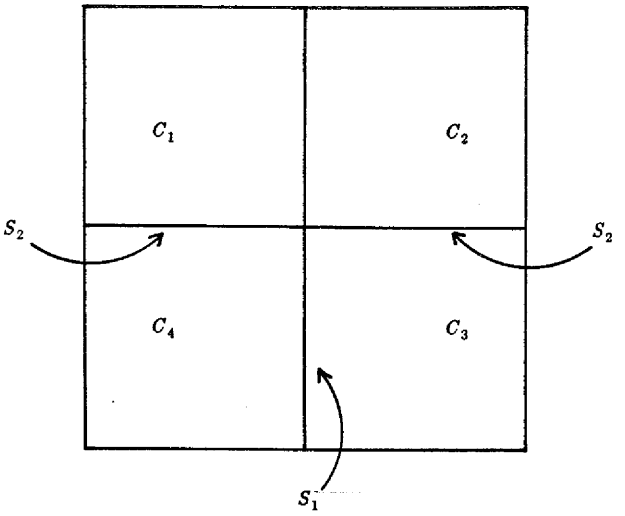


Figure 5.2.1 The first two levels of width-1 dissection on an n by n grid.

horizontal grid-line containing $\frac{1}{2}(n-1)$ grid points is chosen to dissect each n by $\frac{1}{2}(n-1)$ grid into two $\frac{1}{2}(n-1)$ by $\frac{1}{2}(n-1)$ grids. Thus, after two levels of dissections, the original n by n grid is dissected into four identical pieces C_1, C_2, C_3 and C_4. Each piece is an $\frac{1}{2}(n-1)$ by $\frac{1}{2}(n-1)$ grid bordered along two sides. An example is shown in Figure 5.2.1. The dissection technique is then applied recursively to C_1, C_2, C_3 and C_4. A width-1 nested dissection ordering on a 15 by 15 grid is shown in Figure 5.2.2.

Denote a level-k separator by S_k. The result below follows from the fact that the dissection technique is applied recursively on an n by n grid, where $n=2^m-1$.

Lemma 5.2.1

For an n by n grid with $n=2^m-1$,

(1) the number of levels of dissection is $2m-1$,

(2) the number of level-k separators is 2^{k-1}, for $1\leq k\leq 2m-1$, and

(3) the number of grid points on a level-k separator is $2^{m-fl(k/2)}-1$, for $1\leq k\leq 2m-1$, where $fl(k/2)=\lfloor k/2 \rfloor$.

□

For simplicity, we use $Reach_G(x)$ to denote $Reach_G(x,\hat{S})$, where $\hat{S}$ contains nodes whose labellings are less than that of x. It is possible to obtain a bound on $|Reach_G(x)|$, for any node x in G. In the following discussion, the smallest labelling in S_k will be denoted by η_k+1.

Lemma 5.2.2

Let $x\in S_k$. Suppose x has labelling η_k+i. Then

$$|Reach_G(x)| + 1 \leq \rho_k 2^{m-fl(k/2)}-i \ ,$$

where $\rho_k=5$ when k is odd and $\rho_k=7$ when k is even.

Proof:

Let $p=|S_k|$. The separator S_k is obtained by dissecting either a p by p grid or a p by $(2p+1)$ grid. The worst possible situation is that the grid is bordered along four sides. The number of nodes on the additional grid lines is therefore at most $(4p+4)$ for the p by p grid or $(6p+6)$ for the p by $(2p+1)$ grid. The result then follows from Lemma 5.2.1, and the fact that k is odd when a p by p subgrid is dissected and even when a p by $(2p+1)$ subgrid is dissected.

□

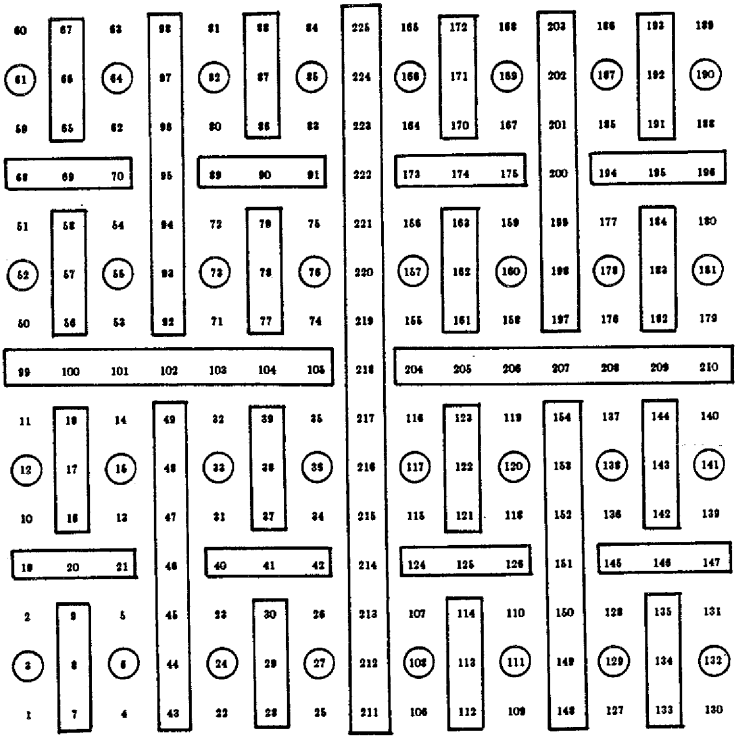


Figure 5.2.2 A width-1 nested dissection ordering on an 15 by 15 finite element grid.

The derivation of $\pi(n,0)$ is similar to that for width-2 nested dissection in Section 4.2 and is given in [23, 36, 65]. We state the result and the proof is omitted.

Lemma 5.2.3

The number of nonzeros in the upper triangular matrix R is

$$\pi(n,0)=\frac{31}{4}n^2\log_2 n + O(n^2) \ , \tag{5.2.1}$$

□

Note that this has the same order of magnitude as that obtained for width-2 nested dissection. The coefficient in the leading term is smaller than that in (4.2.13). Moreover the number of nonzeros in R is optimal (in the order of magnitude sense) as a result of the discussion in Section 4.2.

We now derive the cost of computing R which is more complicated. Here $\hat{G}=(\hat{X},\hat{E})$ will denote the union of the row graphs of those rows that have been eliminated. The graph $G(C_l \bigcup Adj(C_l))$ is denoted by G_l, $1 \le l \le 4$. The following remarks are useful.

(1) As we have pointed in Section 4.2, if $\hat{\mathbb{S}}$ is the elimination sequence of the current row, then the cost of eliminating this row is

$$\sum_{s\in\hat{\mathbb{S}}} \{ \, | \, Reach_{\hat{G}}(x_s,\hat{S}_s) | \, + 1\} \ ,$$

where x_s is the node having labelling s and $\hat{S}_s=\{x_i \in \hat{X} \, | \, i<s\}$. Thus the cost of eliminating the rows of A is

$$\sum_{all\ rows} \sum_{s\in\hat{\mathbb{S}}} \{ \, | \, Reach_{\hat{G}}(x_s,\hat{S}_s) | \, + 1\} \ .$$

(2) Note that the four rows associated with each small square have the same last subscript. So we can assume that they are eliminated together. Furthermore the elimination sequence of the fourth row is maximal and the elimination sequences of the first three rows are subset of that of the fourth one. This is a consequence of the discussion in Section 3.4.

(3) The row ordering strategy is such that the rows associated with $G(C_l)$, $1 \leq l \leq 4$, are eliminated first, those associated with $G(S_2 \bigcup Adj(S_2))$ next, then those associated with $G(S_1 \bigcup Adj(S_1))$ are eliminated last.

(4) We assume that the same column and row ordering strategies are applied to each C_l, $1 \leq l \leq 4$. We can then assume that the cost of eliminating the rows in $G(C_p)$ is the same as that of eliminating the rows in $G(C_q)$, $1 \leq p,q \leq 4$. Let S_2^l and S_2^r denote respectively the left and right level-2 width-1 separators in Figure 5.2.1. If we assume that the nodes of S_2^l and S_2^r, and the rows in $G(S_2^l \bigcup Adj(S_2^l))$ and $G(S_2^r \bigcup Adj(S_2^r))$ are labelled using the same strategies, then we can also assume that the cost of eliminating the rows in $G(S_2^l \bigcup Adj(S_2^l))$ is the same as that of eliminating the rows in $G(S_2^r \bigcup Adj(S_2^r))$.

(5) Consider the rows in $G(C_l)$, $1 \leq l \leq 4$. Before the rows in $G(S_2 \bigcup Adj(S_2))$ and $G(S_1 \bigcup Adj(S_1))$ are eliminated, $G(C_p)$ and $G(C_q)$ are disjoint, for $1 \leq p,q \leq 4$. Note that $G(C_l)$ is simply the graph of an unbordered $\frac{1}{2}(n-1)$ by $\frac{1}{2}(n-1)$ grid. Thus the cost of eliminating the rows of $G(C_l)$ is given by $\theta(\frac{n-1}{2},0)$.

(6) Using remarks (4) and (5), the cost of eliminating the rows of A is therefore given by

$$\theta(n,0) = 4\theta(\frac{n-1}{2},0) + 2\theta_2 + \theta_1 \ ,$$

where θ_i is the cost of eliminating the rows in $G(S_i \bigcup Adj(S_i))$, $i=1,2$.

(7) Consider the elimination of the rows in $G(S_2 \bigcup Adj(S_2))$. We will assume that the grid points in S_2 are labelled consecutively. (Note that the rows in $G(S_1 \bigcup Adj(S_1))$ have not been processed yet.) Recall that η_2+1 is the smallest labelling in S_2. Suppose the last subscript of the row we are eliminating is η_2+j, $2 \leq j \leq \frac{n-1}{2} = 2^{m-1}-1$. Note that the node corresponding to the leading subscript (and the first member in the elimination sequence) is the only node in S_{2m-1} (see Figure 5.2.2). Moreover, using Lemma 3.3.4, the elimination sequence contains $\{\eta_2+i \mid 1 \leq i \leq j\}$ and exactly one level-k separator S_k, for $3 \leq k \leq 2m-1$.

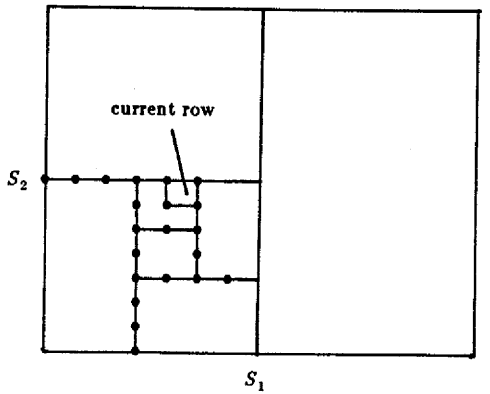


Figure 5.2.3 Darkened nodes are in the elimination sequence of a row in $G(S_2 \bigcup Adj(S_2))$.

This is illustrated in Figure 5.2.3. Using Lemma 5.2.2, the cost of eliminating this row is given by

$$\theta_{2j} = \sum_{k=3}^{2m-1} \sum_{i=1}^{2^{m-fl(k/2)}-1} \left\{ \rho_k 2^{m-fl(k/2)} - i \right\} + \sum_{i=1}^{j} i$$

$$= \frac{49}{24} 2^{2m} - \frac{31}{4} 2^m + \frac{22}{3} + \frac{1}{2} j(j+1) \ . \qquad (5.2.2)$$

Note that there are two small squares which have $\eta_2 + j$ as the last subscript, and there are four rows associated with each small square. So the total cost of eliminating the rows in $G(S_2 \bigcup Adj(S_2))$ is bounded by

$$\theta_2 = 8 \sum_{j=2}^{2^{m-1}-1} \theta_{2j} \ .$$

$$= \frac{25}{3}2^{3m} - \frac{191}{3}2^{2m} + \frac{458}{3}2^{m} - \frac{376}{3} \quad . \tag{5.2.3}$$

(8) The cost of eliminating the rows in $G(S_1 \bigcup Adj(S_1))$ can be derived in a similar manner. Let η_1+1 be the smallest labelling in S_1. Suppose the last subscript of the row we are eliminating is η_1+j, $2 \leq j \leq n$. The node corresponding to the leading subscript is in the last level of dissection. That is, it is in S_{2m-1}. The elimination sequence, using Lemma 3.3.4, contains $\{\eta_1+i \mid 1 \leq i \leq j\}$ and exactly one level-k separator S_k, for $2 \leq k \leq 2m-1$. This is illustrated in Figure 5.2.4. Using Lemma 5.2.2, the cost of eliminating this row is bounded by

$$\theta_{1j} = \sum_{k=2}^{2m-1} \sum_{i=1}^{2^{m-fl(k/2)}-1} \left\{ \rho_k 2^{m-fl(k/2)} - i \right\} + \sum_{i=1}^{j} i$$

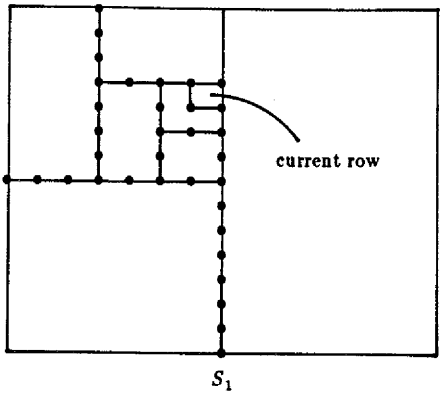


Figure 5.2.4 Darkened nodes are in the elimination sequence of a row in $G(S_1 \bigcup Adj(S_1))$.

$$= \frac{11}{3}2^{2m} - 11(2^m) + \frac{22}{3} + \frac{1}{2}j(j+1) \quad . \tag{5.2.4}$$

Since there are eight rows that have the same last subscript $\eta_1 + j$, the total cost of eliminating the rows in $G(S_1 \bigcup Adj(S_1))$ is bounded by

$$\theta_1 = 8 \sum_{j=2}^{2^m-1} \theta_{1j} \quad .$$

$$= \frac{92}{3}2^{3m} - \frac{440}{3}2^{2m} + \frac{700}{3}2^m - \frac{376}{3} \quad . \tag{5.2.5}$$

Hence using observations (6), (7) and (8), the cost of eliminating all the rows in A is given by

$$\theta(n,0) = 4\theta(\frac{n-1}{2},0) + 2\theta_2 + \theta_1 \quad ,$$

or

$$\theta(m,0) = 4\theta(m-1,0) + 2\theta_2 + \theta_1 \quad .$$

Using (5.2.3) and (5.2.4), the cost is then given by

$$\theta(m,0) = 4\theta(m-1,0) + 2(\frac{25}{3}2^{3m} - \frac{191}{3}2^{2m} + \frac{458}{3}2^m - \frac{376}{3})$$
$$+ (\frac{92}{3}2^{3m} - \frac{440}{3}2^{2m} + \frac{700}{3}2^m - \frac{376}{3})$$
$$= 4\theta(m-1,0) + \frac{142}{3}2^{3m} - 274(2^{2m}) + \frac{1616}{3}2^m - 376 \quad . \tag{5.2.6}$$

Using Lemma 4.1.2(c), the closed form for $\theta(m,0)$ is given by

$$\theta(m,0) = \frac{284}{3}2^{3m} + O(m2^{2m}) \quad . \tag{5.2.7}$$

Replacing m and 2^m by $\log_2(n+1)$ and $(n+1)$ respectively, and expanding $\log_2(n+1)$, we have

$$\theta(n,0) = \frac{284}{3}n^3 + O(n^2 \log_2 n) \quad . \tag{5.2.8}$$

Thus the cost of eliminating the rows for the model problem is also $O(n^3)$, which is the same as

that for width-2 nested dissection. The coefficient obtained may be too large since the bounds obtained in Lemma 5.2.2 may not be tight for some separators. A more careful, but long and tedious, analysis will show that a tighter bound is $\frac{1419}{56}n^3 + O(n^2\log_2 n)$.

We summarize our results in the following theorem.

Theorem 5.2.4

Let A be the matrix associated with an n by n grid and let R be the n^2 by n^2 upper triangular matrix obtained by reducing the rows of A using rotations. If the labelling of the grid points is a width-1 nested dissection labelling and if the row ordering induced by this width-1 nested dissection is used, then the number of nonzeros in R is $O(n^2\log_2 n)$ and the cost of computing R is $O(n^3)$.

□

We now provide some numerical experiments on n by n grids. The objectives are to demonstrate the effectiveness of the row and column orderings induced by width-1 nested dissection and to compare the results with those obtained in Section 4.2.

The column ordering used was the one induced by width-1 nested dissection. The row orderings were the following.

(a) Row ordering A -- This is the row ordering induced by width-1 nested dissection. The rows are arranged so that the last subscripts are in ascending order. (That is, the rows associated with $G(S \bigcup Adj(S))$ are eliminated *last*.)

(b) Row ordering B -- The rows are arranged so that the last subscripts are in descending order. (That is, the rows associated with $G(S \bigcup Adj(S))$ are eliminated *first*.)

(c) Row ordering C -- This is the so-called natural row ordering. The rows are collected from the small squares in the grid row by row.

The following notation is used in Tables 5.2.1 and 5.2.2.

N – number of columns ($N=n^2$).

M – number of rows ($M=4(n-1)^2$).

NZ – number of nonzeros in A.

S – number of storage locations required for the upper triangular matrix R.

τ_A – transformation time for row ordering A (in seconds).

τ_B – transformation time for row ordering B (in seconds).

τ_C – transformation time for row ordering C (in seconds).

The results in Table 5.2.1 confirm that if the column and row orderings induced by width-1 nested dissection are used, the storage requirement and transformation time are $O(n^2\log_2 n)$ and $O(n^3)$ respectively.

The results in Table 5.2.2 show that the row ordering induced by width-1 nested dissection (that is, row ordering A) is indeed better than the other two row orderings used.

Furthermore if we compare the numerical results in Table 5.2.1 with those in Table 4.2.1, we see that the orderings induced by width-1 nested dissection are better than those induced by width-2 nested dissection, in the sense that both the storage requirements and execution times are smaller for width-1 nested dissection.

5.3. Generalized width-1 nested dissection

Complexity results similar to those of Section 5.2 can be derived for finite element problems (see Section 3.5). Suppose $G=(X,E)$ is a finite element graph with $|X|=n$. Lipton, Rose and Tarjan have shown that one can find a width-1 nested dissection ordering for X [56].

Assume that the width-1 nested dissection ordering is used. Let the rows be arranged so that the last subscripts are in ascending ordering. If R is the n by n upper triangular matrix obtained in Algorithm 2.3.1, then it can be shown that the number of nonzeros in R and the cost

n	N	M	NZ	S	$\frac{S}{n^2\log_2 n}$	τ_A	$\frac{\tau_A}{n^3}$
10	100	324	1296	2202	6.63	1.293	0.00129
12	144	484	1936	3315	6.42	2.163	0.00125
14	196	676	2704	4896	6.56	3.673	0.00134
16	256	900	3600	6600	6.45	5.346	0.00131
18	324	1156	4624	8667	6.41	8.159	0.00140
20	400	1444	5776	11123	6.43	11.473	0.00143
22	484	1764	7056	14119	6.54	15.599	0.00146
24	576	2116	8464	17018	6.44	19.825	0.00143
26	676	2500	10000	20859	6.56	26.168	0.00149
28	784	2916	11664	24734	6.56	31.981	0.00146
30	900	3364	13456	29030	6.57	40.798	0.00151

Table 5.2.1 Storage requirement and execution time for width-1 nested dissection orderings.

n	N	M	NZ	τ_A	τ_B	τ_C
10	100	324	1296	1.293	1.593	1.583
12	144	484	1936	2.163	3.060	2.836
14	196	676	2704	3.673	5.480	5.153
16	256	900	3600	5.346	9.229	7.843
18	324	1156	4624	8.159	15.426	12.743
20	400	1444	5776	11.473	22.972	18.775
22	484	1764	7056	15.599	33.121	26.738
24	576	2116	8464	19.825	47.490	36.474
26	676	2500	10000	26.168	64.796	50.040
28	784	2916	11664	31.981	89.048	64.635
30	900	3364	13456	40.798	118.412	84.621

Table 5.2.2 Execution times for three different row orderings.

of computing R are $O(n^{\frac{3}{2}})$ and $O(n\log n)$ respectively. The proofs are similar to those of Section 5.2 and [40, 56]. As in the case of width-2 nested dissection, the results are consistent with those of the model problem.

5.4. Automatic width-1 nested dissection

For general sparse problems, a width-1 nested dissection ordering can be generated automatically using the technique described in Section 4.4. That is, a width-1 separator can be chosen from a level structure of the graph of A^TA rooted at a pseudo-peripheral node. Here each interior level of the level structure is a width-1 separator (even though it may not be minimal). The bound provided by Theorem 4.1.7 on the number of off-diagonal nonzeros in the upper triangular matrix depends on the size of the width-1 separators. Thus instead of using an entire level as a width-1 separator, one can choose a subset from that level so that it is a minimal width-1 separator. The discussion is summarized in Algorithm 5.4.1. We assume that the graph of A^TA, denoted by $G=(X,E)$, is connected.

Algorithm 5.4.1

(1) Find a pseudo-peripheral node r in G using the algorithm from [32].

(2) Generate a level structure rooted at r, $L(r)=\{L_0(r),L_1(r),\cdots,L_{l_r}(r)\}$.

(3) If $l\geq 2$, then choose a minimal width-1 separator from level $\left\lfloor \frac{l_r}{2} \right\rfloor$. Otherwise the whole set X is a width-1 separator.

An algorithm that uses Algorithm 5.4.1 to generate width-1 separators was proposed by George and Liu for generating an automatic width-1 nested dissection ordering of sparse symmetric matrices arising from irregular finite element meshes [28].

We now present some numerical experiments to demonstrate the effectiveness of the column and row orderings induced by the automatic width-1 nested dissection. The test set is the

problem	N	M	NZ	S	τ_A	τ_B	τ_C	τ_D
1	85	219	438	1703	0.553	0.760	0.740	0.743
2	292	958	1916	7572	5.146	7.823	7.000	6.970
3	104	331	662	2166	0.900	1.343	1.133	1.193
4	188	608	1216	4629	2.513	3.946	3.620	3.706
5	176	313	1557	4451	1.497	1.537	1.570	1.787
6	320	1033	4732	6359	5.480	5.110	5.806	5.283
7	320	1033	4719	6359	5.523	4.990	5.800	5.646
8	712	1850	8755	20053	37.521	30.101	43.084	42.384
9	712	1850	8636	20391	38.887	30.468	44.061	44.111
10	225	784	3136	5809	5.026	7.826	6.356	6.390
11	400	1444	5776	11613	13.299	25.415	19.075	19.129
12	402	1512	7152	10267	9.186	15.992	13.202	22.349
13	784	1488	7040	23414	13.776	24.342	15.596	32.681
14	269	900	4208	10302	12.276	17.676	21.182	16.639

Table 5.4.1 Storage requirement in width-1 nested dissection column ordering, and execution times for four different row orderings.

same as the one used in Section 4.4. The column ordering was the one where width-1 separators were generated by the heuristic algorithm described above. Four row orderings were investigated.

(a) Row ordering A -- The rows are arranged so that the last subscripts are in ascending order. This is the row ordering induced by width-2 nested dissection.

(b) Row ordering B -- The rows are arranged so that the last subscripts are in descending order.

(c) Row ordering C -- This is the initial row ordering provided.

(d) Row ordering D -- The rows are arranged so that the numbers of nonzeros in the rows are in ascending order.

The following notation is used in Table 5.4.1.

N – number of columns,

M – number of rows,

NZ – number of nonzeros,

S – number of storage locations required for the upper triangular matrix R,

τ_A – transformation time for row ordering A (in seconds),

τ_B – transformation time for row ordering B (in seconds),

τ_C – transformation time for row ordering C (in seconds),

τ_D – transformation time for row ordering D (in seconds).

Table 5.4.1 contains the results of the experiments. Following are a few remarks on the numerical results in Table 5.4.1.

(1) As in width-2 nested dissection, row ordering D is not a good row ordering in general. The transformation times may be large.

(2) Except for problems 6, 7, 8 and 9, the results show that ordering A (the one induced by width-1 nested dissection) is better than the other three row orderings, in terms of execution time.

(3) Except for problems 1 and 5, the width-1 nested dissection column ordering is better than the width-2 nested dissection column ordering, in terms of storage requirements. In particular, there is a more than 50% reduction in storage requirement for each of problems 6, 7, 8 and 9. This can be explained by the fact that, in automatic width-1 nested dissection, a separator is chosen from a single (middle) level in a rooted level structure, while in automatic width-2 nested dissection, a separator is chosen from two consecutive (middle) levels. Thus the size of a width-1 separator should, in general, be expected to be smaller than that of a width-2 separator. Since the number of nonzeros in the upper

triangular matrix depends on the sizes of the separators, the storage requirement for width-1 nested dissection should be smaller than that for width-2 nested dissection. Table 5.4.2 contains the size of the level-1 width-1 separator for each of the fourteen problems. Notice that each width-1 separator is smaller than the corresponding level-1 width-2 separator in Table 4.4.3.

(4) In terms of execution times, width-1 nested dissection with the induced row ordering is better than width-2 nested dissection with its induced row ordering (except for problems 5 and 11). The reductions in time for problems 6, 7, 8 and 9 are large. For each of problems 5 and 11, the execution time in width-2 nested dissection is only slightly larger than that in width-1 nested dissection.

problem	N	size of separator
1	85	10
2	292	10
3	104	6
4	188	13
5	176	6
6	320	18
7	320	18
8	712	21
9	712	21
10	225	15
11	400	21
12	402	18
13	784	24
14	269	19

Table 5.4.2 Size of width-1 separators in the first level of dissection.

(5) Note that the transformation time for problems 6, 7, 8 and 9 are not the smallest when the induced row ordering is used. (In fact, the transformation times are the smallest when row ordering B is used.) A possible explanation is the following. Consider problem 6 and suppose C is component set obtained in some stage in the automatic width-1 nested dissection. It has been observed that, in most cases, C is dissected into two portions where one portion is very small compared to the size of the other. The number of levels of dissection tend to be large. Recall that, in width-1 nested dissection, the cost of eliminating a row depends in part on the number of levels of dissection and on the size of the separators. Consequently, the *length* of some elimination sequences would tend to be long, and the elimination of the corresponding rows would be expensive. Problems 7, 8 and 9 have the same behavior.

(6) The actual number of nonzeros produced in the transformation process is identical to the number predicted by the symbolic factorization process, except for problems 6 and 8. For each of problems 6 and 8, the actual number is very close to the predicted number (more than 99.5%).

5.5. Minimum degree and width-1 nested dissection orderings

Let A be an m by n sparse matrix with $m \geq n$ and R be the n by n upper triangular matrix obtained in the orthogonal decomposition of A. We have seen that, for the problems we have tested, width-1 nested dissection orderings are better than width-2 nested dissection orderings in the sense that both the amount of space required to store R and the cost of computing R are smaller for most problems. However, for general sparse problems, the storage requirements may be large even for width-1 nested dissection column orderings. The main problem here is that the number of nonzeros in R depends on the size of the separators (see Theorem 4.1.7). For problems arising from grid-like structures, the separators are usually small. For example, if the graph of A^TA is a two-dimensional finite element graph, then it is possible to find a width-1 or width-2 separator that has $O(\sqrt{n})$ nodes (see Section 3.5). For general sparse

problems, even though there are heuristic algorithms for finding separators, there is no guarantee that the separators obtained are small.

For general sparse symmetric positive definite matrices, the minimum degree algorithm [64] usually produces an ordering for which the amount of fill-in in the Cholesky factor is small. Since we are working with the structure of the symmetric matrix A^TA, we can also apply the minimum degree algorithm to A^TA and find a minimum degree ordering (i.e., a minimum degree column ordering for A). For completeness we now describe the basic minimum degree algorithm using our terminology.

Let $G=(X,E)$ be the unlabelled graph of A^TA. In the following discussion, x_i is the node having labelling i, and $S_k=\{x_i \mid i<k\}$, for $1\leq k\leq n$.

(1) Set $S_0=\emptyset$.

(2) For $k=0,1,2,\cdots,n-1$, do the following:

(2.1) Let u be a node such that

$$|Reach_G(u,S_k)| = \min_{x\in X-S_k} |Reach_G(x,S_k)| \ .$$

(u is a node with minimum degree.)

(2.2) Label u next. That is, set $x_{k+1}=u$.

(2.3) Set $S_{k+1}=S_k\bigcup\{x_{k+1}\}$.

The minimum degree algorithm is simple. In each step of the Cholesky decomposition of A^TA, a diagonal element whose corresponding column and row contain the minimum number of nonzeros in the partially reduced (or decomposed) matrix is chosen as the pivot element. Even though this algorithm is heuristic, experience has shown that it is very effective in reducing fill-in for general sparse problems. Much research has been done on the efficient implementation of this algorithm. (See [33, 34] for details.) However few results are known about the behavior of minimum degree algorithms, or the quality of the orderings produced by such algorithms.

Suppose the columns of an m by n matrix A have been ordered by a minimum degree algorithm. In the orthogonal transformation of A, one would like to find a row ordering such that the cost of computing the upper trapezoidal form is small. The results we obtained in Chapter 3 indicate that the choice of a "good" row ordering depends in part on the column ordering. Because of the lack of results on the minimum degree orderings, it appears difficult to identify such a row ordering. The following observation was made in [25]. Suppose the column ordering of A is a minimum degree ordering. If the rows of the (column permuted) matrix are arranged so that the last subscripts are in ascending order, then for certain problems, the cost of computing the upper trapezoidal form would be smaller than the cost when other row orderings are used. In this section, we investigate this problem further and provide some interesting empirical results.

Let B be a sparse symmetric positive definite matrix and $G=(X,E)$ be the graph of B. We assume G is connected. One interesting, and perhaps important, observation about minimum degree orderings is the following. It has been observed that, in many instances, if the node ordering is a minimum degree ordering, then there exists t such that $G(S_t)$ is disconnected but $G(S_k)$ is connected for $t+1 \leq k \leq n$. Here $S_i=\{x_j \in X \mid j<i\}$ and x_j is the node having labelling j. What this means is that the set $T_X=\{x_{t+1}, x_{t+2}, \cdots, x_n\}$ is a width-1 separator in the graph G (even though it may not be minimal). It has also been found that, for such instances, the components of $G(S_t)$ exhibit similar behavior. That is, if $G(C)$ is a component of $G(S_t)$, then there exists a set of nodes $T_C \subseteq C$ which forms a width-1 separator. The labellings of the nodes of T_C are larger than those of the remaining nodes of C.

These observations suggest that a minimum degree ordering could be a width-1 nested dissection ordering. If this is true, then we can immediately conclude that an induced "good" row ordering for a minimum degree column ordering is characterized in the same way as in width-1 nested dissection. That is, the rows are arranged so that *the last subscripts are in ascending order.*

problem	$\|C\|$	$\|T_C\|$	$\|C\|$	$\|T_C\|$	$\|C\|$	$\|T_C\|$	$\|C\|$	$\|T_C\|$
1	85	8	59	3	55	3	29	2
2	292	14	231	6	216	5	163	6
3	104	14	75	4	64	5	55	3
4	188	18	65	4	62	3	57	5
5	175	13	90	3	86	13	81	4
6	320	8	309	1	239	3	233	4
7	320	8	309	1	239	3	233	4
8	712	5	704	26	345	5	298	2
9	712	6	705	21	680	7	305	2
10	225	25	141	7	94	7	51	6
11	400	24	311	14	278	11	177	11
12	402	18	196	2	192	4	154	4
13	784	14	614	8	550	6	354	6
14	269	20	247	18	153	2	150	12

Table 5.5.1 Sizes of some of the width-1 separators induced by a minimum degree ordering.
(C is a component obtained in some level of dissection, T_C is a separator in C.)

We have carried out the following experiment on each of our fourteen test problems. Let A denote the coefficient matrix in each problem. The columns of A are ordered by a minimum degree algorithm. For each component in the labelled graph of A^TA, say $G(C)$, we try to identify the set T_C. If there is such a set T_C, then we remove it from C and repeat the same process recursively on the components in $G(C-T_C)$.

Since the volume of data generated is large, we only present some of the empirical results in Table 5.5.1. A minimum degree ordering on a 15 by 15 finite element grid is displayed in Figure 5.1.1. The induced width-1 separators are shown by darkened lines.

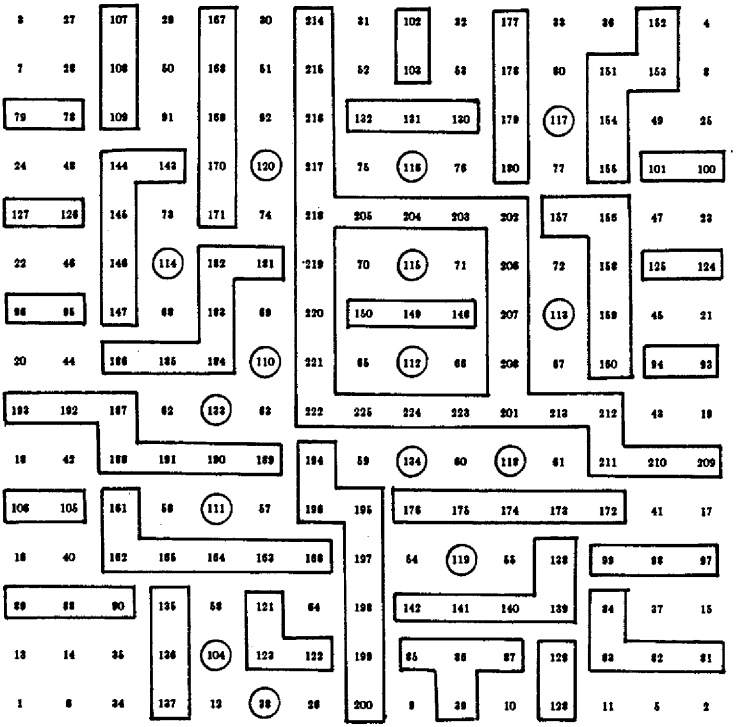


Figure 5.5.1 A minimum degree ordering on an 15 by 15 finite element grid.

The empirical results confirm, at least for our set of test problems, that a minimum degree ordering is indeed a width-1 nested dissection ordering. In Table 5.5.2, we have provided some numerical experiments on the fourteen test problems. The column ordering was a minimum degree ordering. The row orderings were those used in Section 5.4. The notation is the same as that used in Table 5.4.1.

Following are some remarks on the results.

(1) In terms of storage requirements, the minimum degree orderings are definitely better than the width-1 nested dissection orderings (except for problem 11). For problem 11, the difference in space requirements for minimum degree and width-1 nested dissection is small.

problem	N	M	NZ	S	τ_A	τ_B	τ_C	τ_D
1	85	219	438	1490	0.403	0.760	0.673	0.653
2	292	958	1916	6264	3.010	10.693	10.576	10.099
3	104	331	662	1995	0.833	1.837	1.663	1.760
4	188	608	1216	3969	1.773	4.290	3.376	3.490
5	176	313	1557	3547	0.883	2.577	1.490	2.097
6	320	1033	4732	6072	3.313	5.450	4.840	4.690
7	320	1033	4719	6072	3.430	5.476	4.800	5.006
8	712	1850	8755	16584	16.836	18.885	20.305	22.089
9	712	1850	8636	16549	16.636	18.912	21.912	23.375
10	225	784	3136	5753	5.073	10.406	7.350	7.370
11	400	1444	5776	11706	13.823	30.611	28.428	27.992
12	402	1512	7152	8656	6.253	12.972	9.376	15.932
13	784	1488	7040	17311	6.106	11.336	8.889	16.046
14	269	900	4208	9539	9.809	21.945	20.362	17.835

Table 5.5.2 Storage requirement in minimum degree column ordering, and execution times for four different row orderings.

(2) The results show that if the column ordering of the coefficient matrix A is a minimum degree ordering, then by arranging the rows so that the last subscripts are in ascending order, the transformation time (τ_A) is indeed smaller than the transformation times when other row orderings are used.

(3) The transformation times required for row ordering A are smaller than those required for the induced row ordering in width-1 or width-2 nested dissection (except for problems 10 and 11).

(4) For most problems, the actual number of nonzeros produced in R is identical to the number predicted by the symbolic factorization process. The exceptions are problems 6, 7, 8 and 9. However, for each of these four problems, the difference is less than 0.8%.

CHAPTER 6

DENSE ROWS AND UPDATING ALGORITHMS

As we have pointed out in previous chapters, the approach we use in reducing a large sparse matrix A to upper trapezoidal form assumes that A^TA is sparse. However there are examples in which this assumption may be violated. That is, A^TA may be dense even though A is sparse. This usually occurs when A has some relatively "dense" rows. In this chapter we look at the effect of dense rows on the sparsity of and the cost of computing the upper trapezoidal form. We present some algorithms for handling these dense rows.

6.1. Effect of dense rows

The method described in Chapter 2 for reducing a large sparse rectangular matrix A to upper trapezoidal form can be implemented efficiently (in terms of storage requirement and execution time) if the matrix A^TA is sparse and can be permuted symmetrically so that its Cholesky factor is sparse. However it is easy to construct examples which do not satisfy these conditions. One such example is given in Figure 6.1.1 in which A is sparse but R is a full upper triangular matrix (and A^TA is a full matrix). If the matrix A in Figure 6.1.1 is n by n, the number of nonzeros in R is $\frac{1}{2}n(n+1)$ and it can be shown that the cost of computing R is $\frac{1}{2}(n^2+5n-4)$, even though A has only $3n-2$ nonzeros. The row ordering given in Figure 6.1.1 is already one which yields the lowest cost. The high cost of computing R and the large number of nonzeros in R are due to the existence of a completely full row. An upper bidiagonal matrix is obtained after the first $(n-1)$ rows are eliminated; but this structure is destroyed when the last row is processed. This example illustrates that even though a matrix A is sparse, the matrix A^TA and the upper trapezoidal form *may* be dense if some of the rows of A are relatively full. We refer to these rows as *dense rows.*

$$A = \begin{bmatrix} \times & \times & & & & & \\ & \times & \times & & & & \\ & & \times & \times & & & \\ & & & \times & \times & & \\ & & & & \times & \times & \\ & & & & & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \end{bmatrix}$$

$$R = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{bmatrix}$$

Figure 6.1.1 An example illustrating the effect of dense rows.

It should be noted that there are also examples in which the matrix A is sparse and has some relatively dense rows, yet the upper trapezoidal form remains sparse, as illustrated by an example in Figure 6.1.2.

Using our model, each row of a matrix A is a complete graph and the structure of the upper triangular matrix R depends on the structure of the graph of A^TA (and the labelling of the vertices). Thus for the example in Figure 6.1.2, the graph of A^TA is a complete graph because A has a full row. Hence, using the graph of A^TA, the model would predict that R is a dense upper triangular matrix regardless of the choice of node labelling. Consequently the storage requirement predicted by the model will be larger than necessary *even though the actual amount of fill-in is small and the cost of computing R remains small.*

On the other hand the method of reducing A to upper trapezoidal form using rotations remains attractive in spite of the deficiencies in the graph model. There are two reasons. First

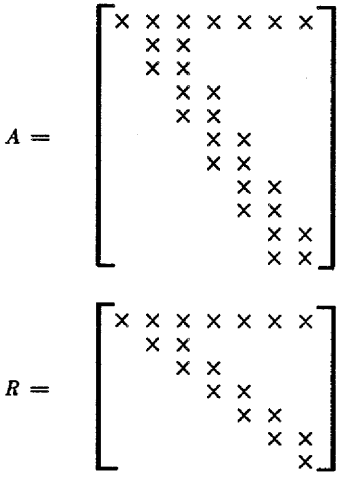


Figure 6.1.2 Another example illustrating the effect of dense rows.

the method provides a numerically stable way for solving systems of linear equations (see the discussion in Sections 1.1 and 1.2). Second it can be implemented efficiently for large sparse A if the structure of A^TA is sparse (see Chapter 2). For problems containing dense rows, it would be desirable if we can arrange the computation so that only an orthogonal decomposition of a sparse matrix with no dense rows is computed. More precisely, suppose $\overline{A}$ denotes the sparse portion of A and assume $\overline{A}^T\overline{A}$ is sparse. An orthogonal decomposition of $\overline{A}$ is then computed. The dense rows are said to be *withheld* from the decomposition. The orthogonal decomposition is used to solve a *modified* problem. Denote the solution by $\overline{x}$. The solution to the original problem is obtained by *updating* $\overline{x}$ *using the previously withheld rows.*

Updating algorithms for least squares problems have been known for a long time even though they were not developed for large sparse problems (see [18, 62]). These algorithms are derived from the normal equations. Recently George and Heath have proposed an updating algorithm for least squares problems which is based on orthogonal decomposition [25]. Heath has proposed another algorithm for least squares problems and has extended it to handle problems with equality constraints [50]. He has also proposed an updating algorithm for partitioned square systems. All these updating algorithms for least squares problems assume that both the sparse portion $\overline{A}$ and the original matrix A are of full rank. However this condition may be too restrictive, as illustrated by the example in Figure 6.1.2. The first row is the one to be withheld; the remaining sparse submatrix has a null column and hence is rank deficient, even though the original matrix may have full column rank. Recently Bjorck has presented a general updating algorithm for solving least squares problems [10], which does not require the sparse submatrix $\overline{A}$ to have full rank. Furthermore it handles both sparse and dense linear constraints. The problem is more complicated if the original matrix A is rank-deficient. Bjorck claims that the algorithm proposed in [10] can be modified to handle this situation.

In this chapter we present some updating algorithms for solving partitioned linear systems. Some of them are modifications of existing ones, but those for solving underdetermined systems are believed to be new.

6.2. Updating algorithms for underdetermined systems

Consider the underdetermined system of linear equations

$$Ax = b \ , \tag{6.2.1}$$

where A is an m by $(n+p)$ matrix and b is an m-vector, with $m \leq n$ and $p \ll n$. We assume that A is partitioned into

$$A = \begin{pmatrix} B & C \end{pmatrix} \ ,$$

where B and C are respectively m by n and m by p matrices. For large problems, B and C

may respectively contain the sparse columns and dense columns of A. It is assumed that BB^T is sparse.

We derive some updating algorithms which are based on computing an orthogonal decomposition of B^T (for example, by applying rotations to the rows of B^T). As far as we know, this is the first attempt in which updating algorithms are employed for solving underdetermined systems.

Case 1 -- Both A and B have full row rank

We assume that an orthogonal decomposition of B^T is given by

$$B^T = Q^T \begin{pmatrix} L^T \\ O \end{pmatrix} ,$$

where Q is an n by n orthogonal matrix, and L is an m by m lower triangular matrix. Since B has full row rank, L must be nonsingular.

The minimal l_2-solution to the underdetermined system (6.2.1) is given by

$$\bar{x} = A^T(AA^T)^{-1}b \quad . \tag{6.2.2}$$

Using the orthogonal decomposition of B, AA^T can be written as

$$\begin{aligned} AA^T &= \begin{pmatrix} B & C \end{pmatrix} \begin{pmatrix} B^T \\ C^T \end{pmatrix} = BB^T + CC^T \\ &= \begin{pmatrix} L & O \end{pmatrix} QQ^T \begin{pmatrix} L^T \\ O \end{pmatrix} + CC^T \\ &= LL^T + CC^T \quad . \end{aligned}$$

Thus the minimal l_2-solution is given by

$$\bar{x} = A^T(LL^T + CC^T)^{-1}b \quad . \tag{6.2.3}$$

Since both A and B have full row rank, the matrices $BB^T{=}LL^T$ and $AA^T{=}LL^T{+}\,CC^T$ are

nonsingular. It is possible to avoid computing the Cholesky decomposition of AA^T (6.2.3) by using the well-known Sherman-Morrison-Woodbury formula which is stated below (see [51, 69, 75]).

Lemma 6.2.1

Let M be an m by m nonsingular matrix, and U and V be m by p matrices. If $M + UV^T$ is nonsingular, then

$$(M + UV^T)^{-1} = M^{-1} - M^{-1}U(I + V^T M^{-1} U)^{-1} V^T M^{-1} \ .$$

□

Applying Lemma 6.2.1 to (6.2.3), the minimal l_2-solution to the underdetermined system (6.2.1) is given by

$$\bar{x} = A^T \{ L^{-T} L^{-1} - L^{-T} L^{-1} C (I + C^T L^{-T} L^{-1} C)^{-1} C^T L^{-T} L^{-1} \} b$$

Even though this expression seems to be very complicated, the resulting computational algorithm which we state below is very simple and straight-forward.

Algorithm 6.2.1

(1) Compute an orthogonal decomposition of B^T. That is, $B^T = Q^T \begin{pmatrix} L^T \\ O \end{pmatrix}$.

(2) Solve the m by m sparse triangular systems $LL^T y = b$.

(3) Solve p sparse triangular systems $LW = C$.

(4) Form the p by p matrix $D = I + W^T W$.

(5) Solve the p by p dense system $Dz = C^T y$.

(6) Solve the m by m sparse triangular systems $LL^T v = Cz$.

(7) Compute $\bar{x} = A^T (y - v)$.

If the lower triangular matrix L is obtained by applying rotations to the rows of B^T (see Algorithm 2.1.2, 2.2.1 or 2.3.1), then Q will be discarded and B^T can be stored on secondary

storage. Also both C and D are small matrices as long as p is small. Thus the amount of storage required to implement Algorithm 6.2.1 is essentially dominated by that required for L. Moreover it is easy to see from the algorithm that the cost of the solution process is essentially given by the cost of computing L (step 1) and the cost of solving the m by m sparse triangular systems (steps 2, 3 and 6).

Case 2 -- A has full row rank, but B is rank-deficient

Assume B has rank r, where $r<m$. We also assume that $(m-r)$ is small. Since B is rank-deficient, it is possible to arrange the columns and rows of B so that the orthogonal decomposition of B^T has the form

$$Q^T\begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix},$$

where Q is an n by n orthogonal matrix, L is an r by r lower triangular matrix and S is an $(m-r)$ by r matrix. In the following discussion we assume that the columns and rows of B (and hence A) have been reordered so that

$$B^T = Q^T\begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix}. \tag{6.2.4}$$

Since A has full row rank, the minimal l_2-solution to the underdetermined system of linear equations (6.2.1) is given by

$$x = A^T(AA^T)^{-1}b \quad .$$

Using the orthogonal decomposition of B, we can write AA^T as

$$AA^T = \begin{pmatrix} B & C \end{pmatrix}\begin{pmatrix} B^T \\ C^T \end{pmatrix} = BB^T + CC^T$$

$$= \begin{pmatrix} L & O \\ S & O \end{pmatrix} QQ^T \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix} + CC^T$$

$$= \begin{pmatrix} L & O \\ S & O \end{pmatrix} \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix} + CC^T \ .$$

Define the m by m lower triangular matrix L_B by

$$L_B = \begin{pmatrix} L & O \\ S & O \end{pmatrix} .$$

Then we have

$$AA^T = L_B L_B^T + CC^T \ .$$

Hence the minimal l_2-solution is given by

$$\bar{x} = A^T (L_B L_B^T + CC^T)^{-1} b \ . \tag{6.2.5}$$

Since L_B is singular, Lemma 6.2.1 cannot be applied to (6.2.5) even though the matrix $AA^T = L_B L_B^T + CC^T$ is nonsingular.

In order to be able to use Lemma 6.2.1, we modify the matrix L_B so that it becomes nonsingular. This process will be referred to as *promoting* the rank of L_B. It is important to note that any modifications should preserve the sparsity of L_B as much as possible. Define the m by m lower triangular matrix $\bar{L}$ by

$$\bar{L} = \begin{pmatrix} L & O \\ S & I \end{pmatrix} .$$

Then

$$\begin{aligned} \bar{L}\bar{L}^T &= \begin{pmatrix} L & O \\ S & I \end{pmatrix} \begin{pmatrix} L^T & S^T \\ O & I \end{pmatrix} = \begin{pmatrix} LL^T & LS^T \\ SL^T & I + SS^T \end{pmatrix} \\ &= \begin{pmatrix} LL^T & LS^T \\ SL^T & SS^T \end{pmatrix} + \begin{pmatrix} O & O \\ O & I \end{pmatrix} = \begin{pmatrix} L & O \\ S & O \end{pmatrix} \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix} + \begin{pmatrix} O & O \\ O & I \end{pmatrix} \\ &= L_B L_B^T + \begin{pmatrix} O & O \\ O & I \end{pmatrix} . \end{aligned}$$

Thus we can write AA^T as

$$AA^T = L_B L_B^T + CC^T = \overline{L}\overline{L}^T - \begin{pmatrix} O & O \\ O & I \end{pmatrix} + CC^T \ .$$

Partition C as

$$C = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} ,$$

where C_1 is r by p and C_2 is $(m-r)$ by p. Define two m by $(p+m-r)$ matrices U and V by

$$U = \begin{pmatrix} C_1 & O \\ C_2 & -I \end{pmatrix} \quad \text{and} \quad V = \begin{pmatrix} C_1 & O \\ C_2 & I \end{pmatrix} .$$

Note that

$$\begin{aligned} UV^T &= \begin{pmatrix} C_1 & O \\ C_2 & -I \end{pmatrix}\begin{pmatrix} C_1^T & C_2^T \\ O & I \end{pmatrix} = \begin{pmatrix} C_1C_1^T & C_1C_2^T \\ C_2C_1^T & C_2C_2^T - I \end{pmatrix} \\ &= \begin{pmatrix} C_1C_1^T & C_1C_2^T \\ C_2C_1^T & C_2C_2^T \end{pmatrix} - \begin{pmatrix} O & O \\ O & I \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}\begin{pmatrix} C_1^T & C_2^T \end{pmatrix} - \begin{pmatrix} O & O \\ O & I \end{pmatrix} \\ &= CC^T - \begin{pmatrix} O & O \\ O & I \end{pmatrix} . \end{aligned}$$

Therefore we have

$$AA^T = L_B L_B^T + CC^T = \overline{L}\overline{L}^T + UV^T \ ,$$

and hence the minimal l_2-solution to the underdetermined system is given by

$$x = A^T(\overline{L}\overline{L}^T + UV^T)^{-1}b \ . \tag{6.2.6}$$

The most important things about this modification are that $\overline{L}$ is now nonsingular and the off-diagonal nonzero structure of $\overline{L}$ is exactly the same as that of L_B. Now we can apply Lemma 6.2.1 to (6.2.6), and the minimal l_2-solution is given by

$$x = A^T\{\overline{L}^{-T}\overline{L}^{-1} - \overline{L}^{-T}\overline{L}^{-1}U(I + V^T\overline{L}^{-T}\overline{L}^{-1}U)^{-1}V^T\overline{L}^{-T}\overline{L}^{-1}\}b \ .$$

We now give the computational algorithm which is similar to Algorithm 6.2.1.

Algorithm 6.2.2

(1) Compute an orthogonal decomposition of B^T. That is, $B^T = Q^T \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix}$. Construct the m by m lower triangular matrix $\bar{L} = \begin{pmatrix} L & O \\ S & I \end{pmatrix}$.

(2) Solve the m by m sparse triangular systems $\bar{L}\bar{L}^T y = b$.

(3) Solve $(p+m-r)$ sparse triangular systems $\bar{L}W_1 = U$ and solve $(p+m-r)$ sparse triangular systems $\bar{L}W_2 = V$.

(4) Form the $(p+m-r)$ by $(p+m-r)$ matrix $D = I + W_2^T W_1$.

(5) Solve the $(p+m-r)$ by $(p+m-r)$ dense system $Dz = V^T y$.

(6) Solve the m by m sparse triangular systems $\bar{L}\bar{L}^T v = Uz$.

(7) Compute $x = A^T(y-v)$.

Apparently we have to compute both W_1 and W_2 in step 3, but in fact we only need to compute *either* W_1 *or* W_2. Let J be the m by $(m-r)$ matrix $\begin{pmatrix} O \\ I \end{pmatrix}$. Then we have $U = \begin{pmatrix} C & -J \end{pmatrix}$ and $V = \begin{pmatrix} C & J \end{pmatrix}$. Thus

$$W_1 = \bar{L}^{-1}U = \begin{pmatrix} \bar{L}^{-1}C & -\bar{L}^{-1}J \end{pmatrix} \quad \text{and} \quad W_2 = \bar{L}^{-1}V = \begin{pmatrix} \bar{L}^{-1}C & \bar{L}^{-1}J \end{pmatrix}.$$

That is, W_1 and W_2 are the same except that the last $(m-r)$ columns have different signs. Moreover it is not difficult to see that $W_2^T W_1$ is the same as $W_2^T W_2$ except that the last $(m-r)$ columns have different signs.

As in case (1), if p and $(m-r)$ are small, the storage requirement will be dominated by the amount of space required for $\bar{L}$, and the cost will depend essentially on steps (1), (2), (3) and (6).

In Section 6.5 we will look at the implementation aspects which include the problem of reordering the rows and columns of B in order to obtain the upper trapezoidal form (6.2.4) and the problem of determining the rank of B.

Case 3 -- Rank-deficient underdetermined systems

The case in which both A and B are rank-deficient seems to be a more difficult problem. The updating technique we have used in the previous cases cannot be used since AA^T is now singular, and the Sherman-Morrison-Woodbury formula is not applicable. However the rank-promotion technique can be modified to solve efficiently a rank-deficient sparse underdetermined system which *does not* possess any dense rows.

Let A be an m by n matrix with $m<n$. Assume A has rank r, where $r<m$. Suppose an orthogonal decomposition of A^T is given by

$$A^T = Q^T \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix},$$

where Q is an n by n orthogonal matrix, L is an r by r lower triangular matrix and S is an $(m-r)$ by r matrix. We have assumed that the rows and columns of A are reordered appropriately.

We now show that the minimal l_2-solution to the underdetermined system (6.2.1) has a form similar to (6.2.2), provided that the system of linear equations is consistent. Let b be partitioned into $\begin{pmatrix} c \\ d \end{pmatrix}$ where c and d are respectively r- and $(m-r)$-vectors. Then the underdetermined system of linear equations can be written as

$$\begin{pmatrix} L & O \\ S & O \end{pmatrix} Qx = \begin{pmatrix} c \\ d \end{pmatrix}.$$

Let Qx be partitioned into $\begin{pmatrix} u \\ v \end{pmatrix}$, where u and v are respectively r- and $(n-r)$-vectors. Then we have

$$\begin{pmatrix} L & O \\ S & O \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix},$$

or

$$\begin{pmatrix} L \\ S \end{pmatrix} u = \begin{pmatrix} c \\ d \end{pmatrix} .$$

That is, $Lu = c$ and $Su = d$. Thus if the system is consistent, we have

$$SL^{-1}c = d \ .$$

Let w be the solution to the r by r sparse triangular systems $LL^T w = c$. Then

$$\bar{x} = A^T \begin{pmatrix} w \\ 0 \end{pmatrix}$$

is a solution to the system (6.2.1), since

$$A\bar{x} = AA^T \begin{pmatrix} w \\ 0 \end{pmatrix} = \begin{pmatrix} L & O \\ S & O \end{pmatrix} QQ^T \begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix} \begin{pmatrix} w \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} L & O \\ S & O \end{pmatrix} \begin{pmatrix} L^T w \\ 0 \end{pmatrix} = \begin{pmatrix} LL^T w \\ SL^T w \end{pmatrix} = \begin{pmatrix} c \\ SL^{-1}c \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} = b \ .$$

Furthermore suppose $\hat{x}$ is any solution to $Ax = b$ and let $\delta = \bar{x} - \hat{x}$. Then

$$A\delta = A(\bar{x} - \hat{x}) = A\bar{x} - A\hat{x} = 0 \ .$$

Note that

$$\| \hat{x} \|_2^2 = \| \bar{x} - \delta \|_2^2 = \| \bar{x} \|_2^2 + \| \delta \|_2^2 - 2\bar{x}^T \delta = \| \bar{x} \|_2^2 + \| \delta \|_2^2 - 2\left(A^T \begin{pmatrix} w \\ 0 \end{pmatrix} \right)^T \delta$$

$$= \| \bar{x} \|_2^2 + \| \delta \|_2^2 - 2\begin{pmatrix} w^T & 0 \end{pmatrix} A\delta = \| \bar{x} \|_2^2 + \| \delta \|_2^2 \geq \| \bar{x} \|_2^2 \ .$$

Thus $\bar{x}$ is in fact the minimal l_2-solution.

In terms of implementation, the approach above is inefficient since we have to identify the matrix L from the orthogonal decomposition. For large sparse problems, the data structure for storing the lower trapezoidal form (L and S) is usually complicated, and it may be difficult and expensive to extract L from the data structure. To solve this problem, we can use the same technique we used in case (2): replace the m by m lower triangular matrix $\begin{pmatrix} L & O \\ S & O \end{pmatrix}$ by

$\bar{L} = \begin{pmatrix} L & O \\ S & I \end{pmatrix}$. Then instead of solving $LL^T w = c$, we do the following. First solve

$$\bar{L}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} L & O \\ S & I \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} .$$

Note that $y_1 = L^{-1}c$ and $y_2 = d - Sy_1 = d - SL^{-1}c$. Because the system is assumed to be consistent, $y_2 = 0$. Next solve

$$\bar{L}^T\begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} L^T & S^T \\ O & I \end{pmatrix}\begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ 0 \end{pmatrix} .$$

Now $z = 0$ and $w = L^{-T}y_1 = L^{-T}L^{-1}c$. Finally the minimal l_2-solution is given by

$$\bar{x} = A^T\begin{pmatrix} w \\ z \end{pmatrix} = A^T\begin{pmatrix} w \\ 0 \end{pmatrix} .$$

Algorithm 6.2.3

(1) Compute an orthogonal decomposition of A^T. That is, $A^T = Q^T\begin{pmatrix} L^T & S^T \\ O & O \end{pmatrix}$. Construct the m by m lower triangular matrix $\bar{L} = \begin{pmatrix} L & O \\ S & I \end{pmatrix}$.

(2) Solve the m by m sparse triangular systems $\bar{L}\bar{L}^T f = b$.

(3) Compute $\bar{x} = A^T f$.

It is important to note that the algorithm only works when the system is consistent.

6.3. Updating algorithms for overdetermined systems

Consider the least squares problem

$$\min_x \| Ax - b \|_2 , \tag{6.3.1}$$

where A is an $(m+p)$ by n matrix and b is an $(m+p)$-vector, with $m \geq n$ and $p \ll m$. We assume that A is partitioned into

$$A = \begin{pmatrix} B \\ C \end{pmatrix},$$

where B and C are respectively m by n and p by n matrices. The vector b is similarly partitioned; that is,

$$b = \begin{pmatrix} e \\ f \end{pmatrix},$$

where e and f are m- and p-vectors respectively. For large problems, B and C may respectively contain the sparse rows and dense rows of A.

This "partitioned" least squares problem also occurs frequently in another context (for example, in statistical computations). Suppose we have solved the least squares problem

$$\min_x \| Bx - e \|_2 . \tag{6.3.2}$$

We may, at a later stage, decide to add more observations (or equations) to form the new least squares problem

$$\min_x \| Ax - b \|_2 .$$

If the problem is large, it is not desirable to compute the new least squares solution by finding the orthogonal decomposition of A directly. It would be much cheaper if one would obtain the new solution by updating the solution to (6.3.2) using C, f and the orthogonal decomposition of B (see [25, 50]). Another alternative is to find the orthogonal decomposition of A indirectly, for example using the orthogonal decomposition of B and C. One such algorithm is described in [13] for small dense problems. For large sparse problems, the latter approach may not be feasible since it is assumed that one can set up the storage scheme for the upper trapezoidal form before carrying out the numerical computation. Since the nonzero structure of the upper trapezoidal form of A may not be the same as that of B, there may not be space in the data structure for the upper trapezoidal form of B to accommodate fill-in when we compute the orthogonal decomposition of A from the orthogonal decomposition of B and the matrix C. Moreover C may

not be available when the least squares problem (6.3.2) is solved. The amount of fill-in caused by the elimination of C cannot be predicted when (6.3.2) is solved.

We now derive some updating algorithms for least squares problems.

Case 1 -- Both A and B have full column rank

It is assumed that an orthogonal decomposition of B is given by

$$B = Q\begin{pmatrix} R \\ O \end{pmatrix},$$

where Q is an m by m orthogonal matrix and R is an n by n upper triangular matrix. Notice that R is nonsingular since B has full column rank.

We first note that the unique least squares solution to (6.3.1) is given by the solution to the system of normal equations

$$\bar{x} = (A^T A)^{-1} A^T b \quad .$$

Using the orthogonal decomposition of B, we have

$$A^T A = \begin{pmatrix} B^T & C^T \end{pmatrix}\begin{pmatrix} B \\ C \end{pmatrix} = B^T B + C^T C$$

$$= \begin{pmatrix} R^T & O \end{pmatrix} Q^T Q \begin{pmatrix} R \\ O \end{pmatrix} + C^T C = R^T R + C^T C \quad .$$

Thus the least squares solution is given by

$$\bar{x} = (R^T R + C^T C)^{-1} A^T b \quad . \tag{6.3.3}$$

This expression is similar to (6.2.3). It needs the inverse of the sum of two matrices, $R^T R$ and $C^T C$. Since both A and B are assumed to have full column rank, $R^T R = B^T B$ and $R^T R + C^T C = A^T A$ are nonsingular. Thus we can use the technique for solving underdetermined systems to compute the least squares solution. That is, we apply the Sherman-Morrison-Woodbury formula to (6.3.3) and obtain:

$$\bar{x} = \{R^{-1}R^{-T} - R^{-1}R^{-T}C^T(I + CR^{-1}R^{-T}C^T)^{-1}CR^{-1}R^{-T}\}A^Tb \quad .$$

We now state the computational algorithm for finding the least squares solution to (6.3.1). This algorithm is similar to Algorithm 6.2.1.

Algorithm 6.3.1

(1) Compute an orthogonal decomposition of B. That is, $B=Q\begin{pmatrix}R\\O\end{pmatrix}$.

(2) Solve the n by n sparse triangular systems $R^TRy=A^Tb$.

(3) Solve p sparse triangular systems $R^TW=C^T$.

(4) Form the p by p matrix $D=I+W^TW$.

(5) Solve the p by p dense system $Dz=Cy$.

(6) Solve the n by n sparse triangular systems $R^TRv=C^Tz$.

(7) Compute $\bar{x}=y-v$.

Note that in step 2, the vector A^Tb can be computed either from A and b, or as $\begin{pmatrix}R^T & O\end{pmatrix}Q^Tb$. Suppose several problems which have the same coefficient matrix are to be solved. Then Q may only be available as a sequence of rotation matrices when the first problem is solved. For subsequent problems, A^Tb must therefore be computed from A and b.

As in the case of solving sparse underdetermined systems with dense columns, the storage requirement is dominated by the amount of space required for R if p is small and Algorithm 2.1.2, 2.2.1 or 2.3.1 is used for computing the upper triangular matrix. The cost of the algorithm depends essentially on the cost of computing R and of solving the sparse triangular systems.

Numerical instability and inaccuracy are the possible difficulties that may occur. It is well known that the accuracy of the computed solution $\bar{x}$ will depend on the square of the condition number of A if the system of normal equations is solved. Thus this updating algorithm may fail to give satisfactory results if A is poorly conditioned. The reason for presenting this

algorithm is that, as in the case of underdetermined systems, it can be adapted very easily to handle the case when B is rank-deficient. This updating algorithm is similar to the one proposed in [62].

Case 2 -- A has full column rank, but B is rank-deficient

As in the case of underdetermined systems, Algorithm 6.3.1 can be adapted easily to find the unique least squares solution to (6.3.1) using the rank-promotion technique when B is rank-deficient. Assume B has rank r, where $r<n$. Suppose the rows and columns of B (and A) have been reordered so that B has the following orthogonal decomposition

$$B = Q\begin{pmatrix} R & S \\ O & O \end{pmatrix} ,$$

where Q is an m by m orthogonal matrix, R is an r by r upper triangular matrix and S is an r by $(n-r)$ matrix.

Since A has full column rank, the unique least squares solution to (6.3.1) is then given by

$$\bar{x} = (A^T A)^{-1} A^T b \quad .$$

Using the orthogonal decomposition of B, we can write $A^T A$ as

$$\begin{aligned} A^T A &= \begin{pmatrix} B^T & C^T \end{pmatrix}\begin{pmatrix} B \\ C \end{pmatrix} = B^T B + C^T C \\ &= \begin{pmatrix} R^T & O \\ S^T & O \end{pmatrix} Q^T Q \begin{pmatrix} R & S \\ O & O \end{pmatrix} + C^T C = \begin{pmatrix} R^T & O \\ S^T & O \end{pmatrix}\begin{pmatrix} R & S \\ O & O \end{pmatrix} + C^T C \\ &= R_B^T R_B + C^T C \quad , \end{aligned}$$

where R_B is the n by n upper triangular matrix $\begin{pmatrix} R & S \\ O & O \end{pmatrix}$. Thus the unique least squares solution is given by

$$\bar{x} = (R_B^T R_B + C^T C)^{-1} A^T b \quad .$$

Now we apply the rank-promotion technique to the singular matrix R_B. Define the n by n upper triangular matrix $\overline{R}$ by

$$\overline{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix} .$$

Then

$$\begin{aligned} \overline{R}^T\overline{R} &= \begin{pmatrix} R^T & O \\ S^T & I \end{pmatrix}\begin{pmatrix} R & S \\ O & I \end{pmatrix} = \begin{pmatrix} R^TR & R^TS \\ S^TR & I + S^TS \end{pmatrix} \\ &= \begin{pmatrix} R^TR & R^TS \\ S^TR & S^TS \end{pmatrix} + \begin{pmatrix} O & O \\ O & I \end{pmatrix} = \begin{pmatrix} R^T & O \\ S^T & O \end{pmatrix}\begin{pmatrix} R & S \\ O & O \end{pmatrix} + \begin{pmatrix} O & O \\ O & I \end{pmatrix} \\ &= R_B^TR_B + \begin{pmatrix} O & O \\ O & I \end{pmatrix} . \end{aligned}$$

Thus we can write A^TA as

$$A^TA = R_B^TR_B + C^TC = \overline{R}^T\overline{R} - \begin{pmatrix} O & O \\ O & I \end{pmatrix} + C^TC .$$

Partition C as

$$C = \begin{pmatrix} C_1 & C_2 \end{pmatrix} ,$$

where C_1 is p by r and C_2 is p by $(n-r)$. Define two $(p+n-r)$ by n matrices U and V by

$$U = \begin{pmatrix} C_1 & C_2 \\ O & -I \end{pmatrix} \qquad \text{and} \qquad V = \begin{pmatrix} C_1 & C_2 \\ O & I \end{pmatrix} .$$

Note that

$$\begin{aligned} U^TV &= \begin{pmatrix} C_1^T & O \\ C_2^T & -I \end{pmatrix}\begin{pmatrix} C_1 & C_2 \\ O & I \end{pmatrix} = \begin{pmatrix} C_1^TC_1 & C_1^TC_2 \\ C_2^TC_1 & C_2^TC_2 - I \end{pmatrix} \\ &= \begin{pmatrix} C_1^TC_1 & C_1^TC_2 \\ C_2^TC_1 & C_2^TC_2 \end{pmatrix} - \begin{pmatrix} O & O \\ O & I \end{pmatrix} = \begin{pmatrix} C_1^T \\ C_2^T \end{pmatrix}\begin{pmatrix} C_1 & C_2 \end{pmatrix} - \begin{pmatrix} O & O \\ O & I \end{pmatrix} \\ &= C^TC - \begin{pmatrix} O & O \\ O & I \end{pmatrix} . \end{aligned}$$

Therefore we have

$$A^TA = \bar{R}^T\bar{R} + U^TV \ ,$$

and the least squares solution is then given by

$$\bar{x} = (\bar{R}^T\bar{R} + U^TV)^{-1}A^Tb \ . \tag{6.3.4}$$

Note that $\bar{R}^T\bar{R}$ and $\bar{R}^T\bar{R} + U^TV$ are nonsingular. Thus we can apply Lemma 6.2.1 to the expression in (6.3.4). In other words, the least squares solution is given by

$$\bar{x} = \{\bar{R}^{-1}\bar{R}^{-T} - \bar{R}^{-1}\bar{R}^{-T}U^T(I + V\bar{R}^{-1}\bar{R}^{-T}U^T)^{-1}V\bar{R}^{-1}\bar{R}^{-T}\}A^Tb \ .$$

The computational algorithm which is stated below is similar to Algorithm 6.2.2.

Algorithm 6.3.2

(1) Compute an orthogonal decomposition of B. That is, $B = Q\begin{pmatrix} R & S \\ O & O \end{pmatrix}$. Construct the n by n upper triangular matrix $\bar{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix}$.

(2) Compute the n-vector $d = A^Tb$.

(3) Solve the n by n sparse triangular systems $\bar{R}^T\bar{R}y = d$.

(4) Solve $(p+n-r)$ sparse triangular systems $\bar{R}^TW_1 = U^T$ and solve $(p+n-r)$ sparse triangular systems $\bar{R}^TW_2 = V^T$.

(5) Form the $(p+n-r)$ by $(p+n-r)$ matrix $D = I + W_2^TW_1$.

(6) Solve the $(p+n-r)$ by $(p+n-r)$ dense system $Dz = Vy$.

(7) Solve the n by n sparse triangular systems $\bar{R}^T\bar{R}v = U^Tz$.

(8) Compute $\bar{x} = y - v$.

As in Algorithm 6.2.2, it is not necessary to compute both W_1 and W_2. Note that

$$W_1 = \overline{R}^{-T} U^T = \overline{R}^{-T} \begin{pmatrix} C_1^T & O \\ C_2^T & -I \end{pmatrix},$$

and

$$W_2 = \overline{R}^{-T} V^T = \overline{R}^{-T} \begin{pmatrix} C_1^T & O \\ C_2^T & I \end{pmatrix}.$$

Thus W_1 and W_2 are identical except that the last $(n-r)$ columns have different signs. Similarly $W_2^T W_1$ is the same as $W_2^T W_2$ except that the last $(n-r)$ columns have different signs.

If p and $(n-r)$ are small, then the storage requirement will be dominated by the space required for $\overline{R}$, and the cost will depend essentially on steps (1), (3), (4) and (7).

Case 3 -- Both A and B are rank-deficient

Update becomes much more complicated when both the original matrix A and the sparse submatrix B are rank-deficient. The approach we used in deriving Algorithms 6.3.1 and 6.3.2 cannot be extended to solve this rank-deficient partitioned least squares problem because $A^T A$ is now singular. Even if C is null (that is, there are no dense rows), there does not exist any algorithm similar to Algorithm 6.2.3 for solving (6.3.1). Note that a rank-deficient least squares problem has an infinite number of solutions. In most applications, we are interested in the one that has the minimal Euclidean norm, the so-called *minimal norm least squares solution.*

Heath presented an algorithm for solving rank-deficient least squares problems in [50]. However it does not handle dense rows. In the same paper another algorithm for solving least squares problems with dense rows is proposed, and it is similar to the one in [25]. Both algorithms requires that A and B have full column rank. Bjorck recently has proposed a general updating algorithm for solving sparse least squares problems in [10] which is similar to those proposed in [50]. It assumes that A has full rank, but B can be rank-deficient. It also allows dense and sparse constraints. All these updating algorithms are derived by considering the residuals in the least squares problem.

We now derive an algorithm for solving a rank-deficient least squares problem with a rank-deficient sparse submatrix B. It is based on an approach that is different from the one used in previous cases, and may be regarded as a generalization of Heath's algorithms. It is a special case of Bjorck's algorithm since we do not include constraints, but on the other hand, it can be considered to be more general because it handles the case in which A is rank-deficient.

Consider the least squares problem

$$\min_x \| Ax - b \|_2 \ , \tag{6.3.5}$$

where $A = \begin{pmatrix} B \\ C \end{pmatrix}$. We assume that A has rank s, where $s \leq n$, and B has rank r, where $r \leq n$. Without loss of generality we assume that the rows and columns of A and B have been reordered so that B has the following orthogonal decomposition

$$B = Q \begin{pmatrix} R & S \\ O & O \end{pmatrix} ,$$

where Q is an m by m orthogonal matrix, R is an r by r upper triangular matrix and S is an r by $(n-r)$ matrix. Let the m-vector $Q^T e$ be partitioned into $\begin{pmatrix} c \\ d \end{pmatrix}$, where c and d are respectively r- and $(m-r)$-vectors.

Let the r-vector w be the solution to the r by r upper triangular system

$$Rw = c \ .$$

For any n-vector x, let ρ denote the square of the residual in the overdetermined system. That is,

$$\rho = \| b - Ax \|_2^2 = \| \begin{pmatrix} e \\ f \end{pmatrix} - \begin{pmatrix} B \\ C \end{pmatrix} x \|_2^2 = \| \begin{pmatrix} e - Bx \\ f - Cx \end{pmatrix} \|_2^2$$

$$= \| e - Bx \|_2^2 + \| f - Cx \|_2^2 \ . \tag{6.3.6}$$

Because the Euclidean norm is invariant under orthogonal transformations, we can write

$$\| e-Bx \|_2^2 = \| Q^T(e-Bx) \|_2^2 = \| \begin{pmatrix} c \\ d \end{pmatrix} - \begin{pmatrix} R & S \\ O & O \end{pmatrix} x \|_2^2 \ .$$

Partition x into $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, where x_1 and x_2 are respectively r- and $(n-r)$-vectors. Then we have

$$\| e-Bx \|_2^2 = \| \begin{pmatrix} c \\ d \end{pmatrix} - \begin{pmatrix} R & S \\ O & O \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \|_2^2 = \| \begin{pmatrix} c-Rx_1-Sx_2 \\ d \end{pmatrix} \|_2^2$$

$$= \| c-Rx_1-Sx_2 \|_2^2 + \| d \|_2^2$$

$$= \| Rw-Rx_1-Sx_2 \|_2^2 + \| d \|_2^2 \ . \tag{6.3.7}$$

Partition C into $\begin{pmatrix} C_1 & C_2 \end{pmatrix}$, where C_1 is p by r and C_2 is p by $(n-r)$. Then

$$\| f-Cx \|_2^2 = \| f - \begin{pmatrix} C_1 & C_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \|_2^2 = \| f-C_1x_1-C_2x_2 \|_2^2 \ . \tag{6.3.8}$$

Using (6.3.7) and (6.3.8), the square of the residual is given by

$$\rho = \| Rw-Rx_1-Sx_2 \|_2^2 + \| f-C_1x_1-C_2x_2 \|_2^2 + \| d \|_2^2 \ . \tag{6.3.9}$$

Let $u=Rw-Rx_1-Sx_2$. Then after rearranging, we have

$$x_1 = w-R^{-1}Sx_2-R^{-1}u \ ,$$

and the second term in (6.3.9) now becomes

$$f-C_1x_1-C_2x_2 = f-C_1w + C_1R^{-1}u-(C_2-C_1R^{-1}S)x_2 \ . \tag{6.3.10}$$

Now define the n by n upper triangular matrix

$$\overline{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix} \ ,$$

and let

$$h = f-C_1w \ .$$

Note that

$$\bar{R}^{-1} = \begin{pmatrix} R^{-1} & -R^{-1}S \\ O & I \end{pmatrix} ,$$

and h can be expressed as

$$h = f - \begin{pmatrix} C_1 & C_2 \end{pmatrix} \begin{pmatrix} w \\ 0 \end{pmatrix} = f - C \begin{pmatrix} w \\ 0 \end{pmatrix} .$$

Now observe that

$$C\bar{R}^{-1} = \begin{pmatrix} C_1 & C_2 \end{pmatrix} \begin{pmatrix} R^{-1} & -R^{-1}S \\ O & I \end{pmatrix} = \begin{pmatrix} C_1R^{-1} & -C_1R^{-1}S + C_2 \end{pmatrix} .$$

Let $V = \begin{pmatrix} V_1 & V_2 \end{pmatrix}$ be the p by n matrix defined by $V_1 = C_1R^{-1}$ and $V_2 = C_2 - C_1R^{-1}S$. That is, $V = C\bar{R}^{-1}$. Then (6.3.10) can be written as

$$f - C_1x_1 - C_2x_2 = h + V_1u - V_2x_2 .$$

Hence we have

$$\rho = \| u \|_2^2 + \| h + V_1u - V_2x_2 \|_2^2 + \| d \|_2^2 .$$

Let $-v = h + V_1u - V_2x_2$; that is,

$$V_1u + v = V_2x_2 - h ,$$

or

$$\begin{pmatrix} V_1 & I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = V_2x_2 - h .$$

Hence the square of the residual becomes

$$\rho = \| u \|_2^2 + \| v \|_2^2 + \| d \|_2^2 = \| \begin{pmatrix} u \\ v \end{pmatrix} \|_2^2 + \| d \|_2^2 .$$

The original problem is to choose x so that ρ is minimized. Note that d is a constant vector. Thus the problem can now be stated as the following: find the r-vector u and the p-

vector v such that $\left\| \begin{pmatrix} u \\ v \end{pmatrix} \right\|_2$ is minimized with u and v satisfying the p by $(r+p)$ underdetermined system of linear equations

$$\begin{pmatrix} V_1 & I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = V_2 x_2 - h \quad . \tag{6.3.11}$$

Notice that the p by $(r+p)$ matrix $\begin{pmatrix} V_1 & I \end{pmatrix}$ is *always* of full rank. Thus if the vector x_2 were known, then the problem would be equivalent to finding the minimal l_2-solution to the full rank p by $(r+p)$ underdetermined system (6.3.11). Since we are assuming p is small, the underdetermined system could be solved using any method for small systems.

Assume x_2 is known for the time being. Suppose the p by $(r+p)$ matrix $\begin{pmatrix} V_1 & I \end{pmatrix}$ has an orthogonal decomposition

$$\begin{pmatrix} V_1^T \\ I \end{pmatrix} = \overline{Q} \begin{pmatrix} L^T \\ O \end{pmatrix} ,$$

where $\overline{Q}$ is an $(r+p)$ by $(r+p)$ orthogonal matrix and L is an p by p lower triangular matrix. Then the underdetermined system can be written as

$$\begin{pmatrix} L & O \end{pmatrix} \overline{Q}^T \begin{pmatrix} u \\ v \end{pmatrix} = V_2 x_2 - h \quad . \tag{6.3.12}$$

Partition the $(r+p)$-vector $\overline{Q}^T \begin{pmatrix} u \\ v \end{pmatrix}$ into $\begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix}$, where $\overline{u}$ and $\overline{v}$ are r- and p-vectors respectively. Then we have

$$\begin{pmatrix} L & O \end{pmatrix} \begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix} = V_2 x_2 - h \quad . \tag{6.3.13}$$

The general solution to the underdetermined system (6.3.13) has the form

$$\overline{u} = L^{-1}(V_2 x_2 - h)$$

$$\overline{v} = \textit{arbitrary} \quad .$$

Thus the general solution to the underdetermined system (6.3.11) is given by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \bar{Q}\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} .$$

However we are interested in finding the minimal l_2-solution to (6.3.11); that is, we want to solve

$$\begin{aligned} \min_{u,v} \left\| \begin{pmatrix} u \\ v \end{pmatrix} \right\|_2^2 &= \min_{\bar{u},\bar{v}} \left\| \bar{Q}\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} \right\|_2^2 = \min_{\bar{u},\bar{v}} \left\| \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} \right\|_2^2 \\ &= \min_{\bar{u},\bar{v}} \{ \| \bar{u} \|_2^2 + \| \bar{v} \|_2^2 \} \\ &= \min_{x_2,\bar{v}} \{ \| L^{-1}(V_2 x_2 - h) \|_2^2 + \| \bar{v} \|_2^2 \} , \end{aligned}$$

since x_2 is not known yet. Obviously the minimal l_2-solution is obtained by choosing $\bar{v}=0$ and x_2 so that $\| L^{-1}V_2 x_2 - L^{-1}h \|_2$ is minimized. That is, x_2 should be chosen to be the solution to the p by $(n-r)$ linear least squares problem

$$\min_{x_2} \| L^{-1}V_2 x_2 - L^{-1}h \|_2 . \tag{6.3.14}$$

Assume $\bar{x}_2$ is a solution to this least squares problem. (We will consider the solution of (6.3.14) later.) The minimal l_2-solution to (6.3.12) is given by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \bar{Q}\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} = \bar{Q}\begin{pmatrix} L^{-1}(V_2\bar{x}_2 - h) \\ 0 \end{pmatrix} , \tag{6.3.15}$$

and a solution to the original problem (6.3.5) is

$$\bar{x} = \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} ,$$

where $\bar{x}_1 = w - R^{-1}u - R^{-1}S\bar{x}_2$, u is given in (6.3.15) and $\bar{x}_2$ is a solution to (6.3.14).

When we considered the solution of rank-deficient underdetermined systems, we pointed out that it may be inefficient to extract R and S from the data structure when the problem is large and sparse. The same comment applies here. We prefer to work with the n by n upper triangular matrix $\bar{R}$ instead of R and S. If we let

$$z = -R^{-1}u - R^{-1}S\overline{x}_2 \ .$$

Then we can write down two systems of linear equations

$$Rz + S\overline{x}_2 = -u$$

and

$$\overline{x}_2 = \overline{x}_2 \ .$$

In matrix notation, we have

$$\begin{pmatrix} R & S \\ O & I \end{pmatrix} \begin{pmatrix} z \\ \overline{x}_2 \end{pmatrix} = \begin{pmatrix} -u \\ \overline{x}_2 \end{pmatrix} .$$

After solving this upper triangular system, $\overline{x}_1$ is simply given by $\overline{x}_1 = w + z$.

The only problem that has to be considered is the solution of the least squares problem in (6.3.14):

$$\min_{x_2} \| L^{-1}V_2 x_2 - L^{-1}h \|_2 \ .$$

Note that $L^{-1}V_2$ is p by $(n-r)$. (There may be two cases: either $p \leq n-r$ or $p \geq n-r$). It should be pointed out that apart from the solution of this least squares problem, each stage in the solution process provides a unique solution. If the original matrix A has full rank, then the solution to the original problem (6.3.5) must be unique. In other words, $L^{-1}V_2$ must have full rank and thus (6.3.14) must have a unique least squares solution. Since both p and $(n-r)$ are assumed to be small, the problem (6.3.14) can be solved using any standard method for small dense problems (for example, see [55]). However if A is rank-deficient, then solving (6.3.14) is the only stage that could produce a non-unique solution. In other words, the matrix $L^{-1}V_2$ must be rank-deficient in this case. Since the problem is small, one may use, for example, a singular value decomposition [45, 47] of $L^{-1}V_2$ to find the minimal norm least squares solution to (6.3.14). However there are two important observations.

(1) The vector u depends only on the residual vector $L^{-1}(V_2\bar{x}_2-h)$ which remains the same for any solution $\bar{x}_2$ to (6.3.14).

(2) Even if $\bar{x}_2$ is the minimal norm least squares solution to (6.3.14), the final solution $\bar{x}=\begin{pmatrix}\bar{x}_1\\ \bar{x}_2\end{pmatrix}$ may not be the minimal norm least squares solution to the original problem (6.3.5). The minimal norm least squares solution to (6.3.5) also depends on R, S and w.

Now assume A is rank-deficient. In order to compute the minimal norm least squares solution to (6.3.5), we consider the following approach. Recall that u is *independent* of the choice of solution to the linear least squares problem (6.3.14) since any solution will produce the same residual. Thus we will not compute $\bar{x}_2$ when we solve (6.3.14). We only compute the residual vector $L^{-1}(V_2\bar{x}_2-h)$ and the vector u. Recall that for any solution $\bar{x}_2$ to (6.3.14), $\bar{x}_1=w-R^{-1}u-R^{-1}S\bar{x}_2$. To find the minimal norm solution to (6.3.5), we have to find $\bar{x}_1$ and $\bar{x}_2$ that solve the following problem

$$\begin{aligned}\min_{\bar{x}_1,\bar{x}_2}\left\|\begin{pmatrix}\bar{x}_1\\ \bar{x}_2\end{pmatrix}\right\|_2 &= \min_{\bar{x}_2}\left\|\begin{pmatrix}w-R^{-1}u-R^{-1}S\bar{x}_2\\ \bar{x}_2\end{pmatrix}\right\|_2\\ &= \min_{\bar{x}_2}\left\|\begin{pmatrix}R^{-1}S\bar{x}_2+R^{-1}u-w\\ -\bar{x}_2\end{pmatrix}\right\|_2\\ &= \min_{\bar{x}_2}\left\|\begin{pmatrix}R^{-1}S\\ -I\end{pmatrix}\bar{x}_2-\begin{pmatrix}w-R^{-1}u\\ 0\end{pmatrix}\right\|_2 .\end{aligned} \tag{6.3.16}$$

This is just an n by $(n-r)$ least squares problem whose coefficient matrix $\begin{pmatrix}R^{-1}S\\ -I\end{pmatrix}$ is *always* of full rank. Apparently one has to go into the data structure to find R and S in order to compute $R^{-1}S$ or $R^{-1}u$. In fact this is not necessary since the n by $(n-r)$ matrix $\begin{pmatrix}R^{-1}S\\ -I\end{pmatrix}$ is the solution to the n by n triangular systems

$$\overline{R}Y = \begin{pmatrix} R & S \\ O & I \end{pmatrix} Y = \begin{pmatrix} O \\ -I \end{pmatrix},$$

where Y is an n by $(n-r)$ matrix. Similarly the vector $\begin{pmatrix} R^{-1}u \\ 0 \end{pmatrix}$ can be obtained by solving

$$\overline{R}\overline{z} = \begin{pmatrix} R & S \\ O & I \end{pmatrix} \overline{z} = \begin{pmatrix} u \\ 0 \end{pmatrix}.$$

Finally the residual vector of (6.3.16) at the solution will be the minimal norm least squares solution to the original problem in (6.3.5).

The discussion above is summarized in the following algorithm.

Algorithm 6.3.3

(1) Compute an orthogonal decomposition of B. That is, $B = Q\begin{pmatrix} R & S \\ O & O \end{pmatrix}$. Construct the n by n upper triangular matrix $\overline{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix}$.

(2) Compute the m-vector $Q^T b$ and partition it into $\begin{pmatrix} c \\ d \end{pmatrix}$, where c and d are r- and $(m-r)$-vectors respectively.

(3) Solve the n by n sparse upper triangular system $\overline{R}y = \begin{pmatrix} c \\ 0 \end{pmatrix}$. Thus $y = \begin{pmatrix} w \\ 0 \end{pmatrix}$.

(4) Compute the p-vector $h = f - Cy$.

(5) Solve p sparse triangular systems $\overline{R}^T V^T = C^T$. Partition $V = \begin{pmatrix} V_1 & V_2 \end{pmatrix}$, where V_1 and V_2 are p by r and p by $(n-r)$ matrices.

(6) Compute the orthogonal decomposition of $\begin{pmatrix} V_1^T \\ I \end{pmatrix}$:

$$\begin{pmatrix} V_1^T \\ I \end{pmatrix} = \overline{Q}\begin{pmatrix} L^T \\ O \end{pmatrix}.$$

(7) Solve $(n-r)$ lower triangular systems $LW = V_2$.

(8) Solve the lower triangular system $Lg = h$.

(9) Compute the p-vector α where $\| \alpha \|_2 = \min_{x_2} \| Wx_2 - g \|_2$.

(10) Compute $\begin{pmatrix} u \\ v \end{pmatrix} = \overline{Q} \begin{pmatrix} \alpha \\ 0 \end{pmatrix}$.

(11) Solve $(n-r)$ sparse triangular systems $\overline{R}Y = \begin{pmatrix} O \\ -I \end{pmatrix}$.

(12) Solve the triangular system $\overline{R}\overline{x} = \begin{pmatrix} u \\ 0 \end{pmatrix}$.

(13) Solve $\min_{x_2} \| Yx_2 - (y - \overline{x}) \|_2$ and let $\overline{x}_2$ be the least squares solution.

(14) The minimal norm solution to (6.3.5) is given by $\overline{x} = Y\overline{x}_2 - (y - \overline{x})$.

Note that this algorithm also handles the cases when A has full rank and B has either full or deficient rank. Of course C can be null. In terms of complexity, Algorithm 6.3.3 is more complicated than either Algorithm 6.3.1 or 6.3.2. When A has full column rank, it is hard to say for sure if Algorithm 6.3.3 is more stable or accurate than Algorithm 6.3.1 or 6.3.2. This problem is currently under investigation. However it is certain that Algorithm 6.3.3 is more general than the other two.

6.4. Updating algorithms for nonsingular square systems

Consider the system of linear equations

$$Ax = b \ , \tag{6.4.1}$$

where A is an n by n nonsingular matrix and b is an n-vector. We assume that A is partitioned into

$$A = \begin{pmatrix} B \\ C \end{pmatrix} ,$$

where B and C are respectively m by n and p by n matrices, with $m+p=n$ and $p \ll m$. The vector b is partitioned in a similar manner; that is,

$$b = \begin{pmatrix} e \\ f \end{pmatrix},$$

where e and f are respectively m- and p-vectors. In the applications we have in mind, B and C would respectively contain the sparse and dense rows of A.

One interesting note about this problem is that when A is square and nonsingular, *all* rows of A are linearly independent. Consequently B must have rank m and adding C to B must restore the rank to n. Heath made use of this observation and proposed an algorithm for solving the partitioned square system using an orthogonal decomposition of B [50].

Alternatively one may regard (6.4.1) as a linear least squares problem. Thus the method used to find the minimal norm least squares solution in Algorithm 6.3.3 can be simplified to compute the unique solution to (6.4.1).

Suppose an orthogonal decomposition of B is given by

$$B = Q \begin{Bmatrix} R & S \end{Bmatrix},$$

where Q is an m by m orthogonal matrix, R is an m by m upper triangular matrix and S is an m by p matrix. Then the linear system can be written as

$$Q^T B x = Q^T e \quad , \tag{6.4.2}$$

and

$$Cx = f \quad . \tag{6.4.3}$$

Let $c=Q^T e$. Partition x into $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, where x_1 and x_2 are respectively m- and p-vectors. Also partition C into $\begin{Bmatrix} C_1 & C_2 \end{Bmatrix}$, where C_1 and C_2 are respectively p by m and p by p matrices. Then (6.4.2) and (6.4.3) become respectively

$$\begin{pmatrix} R & S \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = Rx_1 + Sx_2 = c \quad , \tag{6.4.4}$$

and

$$\begin{pmatrix} C_1 & C_2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = C_1x_1 + C_2x_2 = f \quad . \tag{6.4.5}$$

Let $Rw=c$. We can write (6.4.4) as

$$x_1 + R^{-1}Sx_2 = w$$

or

$$x_1 = w - R^{-1}Sx_2 \quad .$$

Substituting x_1 in (6.4.5), we have

$$C_1(w - R^{-1}Sx_2) + C_2x_2 = f \quad .$$

After rearranging, x_2 is then given by

$$(C_2 - C_1R^{-1}S)x_2 = f - C_1w = f - \begin{pmatrix} C_1 & C_2 \end{pmatrix}\begin{pmatrix} w \\ 0 \end{pmatrix} \quad .$$

Let $\overline{R}$ be the n by n upper triangular matrix defined by

$$\overline{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix} \quad .$$

The p by p matrix $C_2 - C_1R^{-1}S$ can be obtained from the last p columns of the matrix

$$C\overline{R}^{-1} = \begin{pmatrix} C_1 & C_2 \end{pmatrix}\begin{pmatrix} R^{-1} & -R^{-1}S \\ O & I \end{pmatrix} = \begin{pmatrix} C_1R^{-1} & -C_1R^{-1}S + C_2 \end{pmatrix} \quad .$$

Let $V=\begin{pmatrix} V_1 & V_2 \end{pmatrix}$ be the p by n matrix defined by $V_1=C_1R^{-1}$ and $V_2=C_2-C_1R^{-1}S$; that is, $V=C\overline{R}^{-1}$. Then x_2 is obtained by solving the p by p system

$$V_2x_2 = f - C\begin{pmatrix} w \\ 0 \end{pmatrix} \quad .$$

Finally x_1 is given by $x_1 = w - R^{-1}Sx_2$, where $-R^{-1}Sx_2$ can be obtained from

$$\begin{pmatrix} -R^{-1}Sx_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} R^{-1} & -R^{-1}S \\ O & I \end{pmatrix} \begin{pmatrix} 0 \\ x_2 \end{pmatrix} = \overline{R}^{-1} \begin{pmatrix} 0 \\ x_2 \end{pmatrix} .$$

For completeness we state the updating algorithm for solving nonsingular partitioned square systems, which is a simplified version of Algorithm 6.3.3.

Algorithm 6.4.1

(1) Compute an orthogonal decomposition of B. That is, $B = Q \begin{pmatrix} R & S \end{pmatrix}$, where R is an m by m upper triangular matrix and S is an m by p matrix. Construct the n by n upper triangular matrix $\overline{R} = \begin{pmatrix} R & S \\ O & I \end{pmatrix}$.

(2) Compute the m-vector $c = Q^T e$.

(3) Solve the n by n sparse upper triangular system $\overline{R}y = \begin{pmatrix} c \\ 0 \end{pmatrix}$. (Thus $y = \begin{pmatrix} w \\ 0 \end{pmatrix}$.)

(4) Compute the p-vector $h = f - Cy$.

(5) Solve the sparse upper triangular systems $\overline{R}^T V^T = C^T$. Partition V into $\begin{pmatrix} V_1 & V_2 \end{pmatrix}$, where V_1 and V_2 are respectively p by m and p by p matrices.

(6) Solve the p by p dense system $V_2 x_2 = h$.

(7) Solve the n by n sparse upper triangular system $\overline{R} \begin{pmatrix} z \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ x_2 \end{pmatrix}$.

(8) Compute the solution $x = y + \begin{pmatrix} z \\ x_2 \end{pmatrix}$.

6.5. Implementation details

There are a few things we should consider in the implementation of the algorithms presented in previous sections.

(1) It has been assumed throughout the discussion that sparsity will be exploited when the orthogonal decomposition of the sparse portion B is computed. This can be achieved by using the method presented in Chapter 2. That is, the orthogonal decomposition is obtained by eliminating the rows of either B (for square and overdetermined systems) or B^T (for underdetermined systems) using rotations. We assume that the rotations are discarded once they are used, that the rows of B or B^T are stored on secondary storage, and that the number of dense rows is small. The amount of main storage required is therefore essentially governed by the sparsity of the upper trapezoidal form of B or B^T.

(2) Suppose B is the m by n matrix whose orthogonal decomposition is to be computed, and assume $m \geq n$. In general the column ordering of B will be chosen so that the upper trapezoidal form is sparse. Thus we do not want to change this ordering during the numerical computation of the orthogonal decomposition, otherwise the effort of finding a good column ordering will be wasted and the amount of fill-in may be large. This means that if B is rank-deficient, we do not want to change the column ordering of B (during the numerical computation) so that the upper trapezoidal form has the form

$$\begin{pmatrix} R & S \\ O & O \end{pmatrix}.$$

Fortunately this is not necessary. All we need is to identify the null rows in the first n rows of the upper trapezoidal form. Then we replace those null rows by the appropriate rows of the identity matrix. More precisely, we replace the diagonal element of a null row by one. (See [50] for details on this technique.) Sometimes, instead of ones, we can put some other numbers on the diagonal in order to decrease the condition number of the upper triangular matrix. However, the problem of deciding what numbers should be used seems to be a difficult one and little work has been done.

(3) Let B be an m by n matrix with $m \geq n$, and R be an n by n upper triangular matrix obtained via an orthogonal decomposition. Assume B has rank r, where $r < n$. A subtle

problem to consider is the identification of the null rows. Since only finite-precision arithmetic is used, it is unlikely that one can find exactly $(n-r)$ null rows in R. More probably, one will find $(n-r)$ rows which contain very small elements. A reliable way to determine the rank of B in finite-precision arithmetic is to use a singular value decomposition (see [45, 47, 55]). However if B is large and sparse, computing a singular value decomposition may be very expensive, both in storage requirement and execution time. A heuristic but often reliable method is to compare the diagonal elements of R with some small tolerance. Suppose there are η diagonal elements that are smaller than this tolerance in magnitude. Then $(n-\eta)$ will be regarded as the *numerical rank* of B. One way of choosing the tolerance is described in [14]. The errors in the matrix A are estimated. The rows and columns of A are scaled so that the error estimates are approximately equal to a common value, say τ. Then τ is used as the tolerance. Experience has shown that the numerical rank obtained is usually the same as the rank of B.

(4) Another subtle problem which is not directly related to the updating algorithms is the identification of the dense portion. We have left the meaning of dense rows somewhat vague. Even though we know a full row is certainly a dense row, some rows which have a relatively small number of nonzeros may also be regarded as dense rows if the rotation of these rows introduces substantial fill-in in the upper trapezoidal form or they are expensive to eliminate. The problem of identifying dense rows seems to be very difficult and no results are available. Following is an heuristic scheme. Suppose Y is a set of rows that contain substantially more nonzeros than the others. One can withhold some of the rows of Y. Experience has shown that, if this scheme is used, the storage requirement could be reduced substantially (compared to the space required when no rows are withheld). Similarly, the execution time, which includes the time for performing update, could also be smaller than the time required when no rows are withheld.

CHAPTER 7

CONCLUDING REMARKS

7.1. Contributions

In this thesis we have studied the problem of computing the QR-decomposition of a sparse m by n matrix, with $m \geq n$. Our approach can be summarized as follows.

(1) The columns of A are permuted so that the upper triangular matrix R obtained in the QR-decomposition is sparse. Denote the permuted matrix by $\overline{A}$.

(2) The rows of $\overline{A}$ are then ordered so that the cost of computing R is small. Denote the reordered matrix by $\hat{A}$.

(3) The rows of $\hat{A}$ are processed one at a time using the algorithm proposed in [25].

The upper triangular matrix R may be dense when A has a few dense rows. In this case we only compute the QR-decomposition of the sparse portion of A.

The major contributions in this thesis can be summarized as follows.

(1) Development of a graph model for studying the row and column ordering problems.

(2) Derivation of graph-theoretic results and relationships between row and column orderings.

(3) Characterization of good row orderings for nested dissection column orderings.

(4) Complexity analyses of nested dissection orderings for a model problem.

(5) Presentation of numerical experiments which show that a minimum degree ordering is a width-1 nested dissection ordering (at least for our set of test problems).

(6) Derivation of algorithms for solving systems of linear equations using the orthogonal decomposition of the sparse portion of A and its dense rows (or dense columns in underdetermined systems).

As we have pointed out in Chapter 2, there are many heuristic algorithms for finding good column orderings for sparse QR-decompositions. On the other hand, few algorithms exist for finding good row orderings. In this thesis we have proposed a graph model to study the row and column ordering problems in a systematic manner. The model is based on the one that is commonly used in sparse Cholesky decomposition. The graph-theoretic results obtained have provided us with a mechanism of constructing "good" row and column orderings.

Using the results, we have proposed a width-2 nested dissection ordering which induces both good row and column orderings, and we showed that the induced row ordering has a simple characterization. Width-2 separators play an important role in finding a width-2 nested dissection ordering. It was shown that both the storage required for storing R and the cost of computing R depend in part on the sizes of the width-2 separators, which means one should use small separators. Only a restricted class of problems is guaranteed to have small width-2 separators, but there is a larger class of problems that is known to have small width-1 separators. Because of this, we would like to replace width-2 separators by width-1 separators. This gives a width-1 nested dissection ordering. We were able to show that a width-1 nested dissection ordering also induces a good row ordering. For problems defined on n by n grids (our model problem), we showed that the storage requirement for the upper triangular matrix and the cost of computing it are respectively $O(n^2 \log_2 n)$ and $O(n^3)$ for both width-1 and width-2 nested dissection orderings, but the coefficients in the leading terms in width-1 nested dissection are smaller than those in width-2 nested dissection. Numerical experiments on various types of problems indicated that width-1 nested dissection orderings are in general better than width-2 nested dissection orderings, both in terms of storage requirement and execution time.

For our set of test problems, we observed that a minimum degree ordering is a width-1 nested dissection ordering. This is important since the amount of fill-in in R is usually small if the columns of A are labelled by a minimum degree algorithm. Assuming that a minimum degree ordering is a width-1 nested dissection ordering, one can immediately characterize a good row

ordering for A. Numerical experiments indicated that the minimum degree column and row orderings are better than both width-1 and width-2 nested dissection orderings, both in terms of storage requirement and execution time.

The numerical experiments suggested that the graph model we proposed is a good model, in the sense that the structure of R determined from the model is very close to the actual structure produced in the numerical computation. For the model problem, it was proved that the structure of R predicted by the graph model is identical to the actual structure.

In the discussion above, we have assumed A^TA is sparse if A is sparse. This is not always true. In fact, when A contains some dense rows, the upper triangular matrix R may be dense. One way to handle this situation is to withhold the dense rows from the QR-decomposition. That is, we only compute the QR-decomposition of the sparse portion of A. We have derived several new algorithms for solving linear systems using the QR-decomposition of the sparse portion of A and the withheld rows. The algorithms are capable of handling the cases in which the sparse portion of A is of full rank or rank-deficient. For overdetermined systems, the matrix A can even be rank-deficient and at the same time have dense rows.

7.2. Further work and open problems

Much work remains to be done in the area related to the problems we have considered in this thesis. Following is a list of some of the open problems.

(1) The empirical results in Section 5.5 indicate that, at least for our set of test problems, a minimum degree ordering is also a width-1 nested dissection ordering. An open problem is either to prove that or to find conditions under which a minimum degree ordering is equivalent to a width-1 nested dissection ordering. This is important because if a minimum degree ordering is indeed a width-1 nested dissection ordering, then we already know how to characterize a good row ordering in the QR-decomposition of a sparse m by n matrix, with $m \geq n$, when the column labelling is a minimum degree labelling. On the other hand, if a minimum degree ordering is not a width-1 nested dissection ordering, then another problem

is to identify a good row ordering.

(2) The graph model we have proposed assumes that exact cancellation does not occur and all elimination sequences are maximal. This is not always true. Consider the following example.

$$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & & \\ & & & \times & \\ & \times & & & \times \\ & & & & \times \\ & & & & \times \end{pmatrix}.$$

Since the first row is a full row, the graph of A^TA is a complete graph, and the symbolic factorization process will predict that the upper triangular matrix R is full. However, R is not full. If R is computed using row elimination, then its actual structure is given by

$$R = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & & \times \\ & & \times & & \times \\ & & & \times & \\ & & & & \times \end{pmatrix}.$$

One way to handle this situation is to withhold the first row and compute the QR-decomposition of the submatrix containing the last five rows of A. Notice that the elimination sequence of the first row is not maximal. In fact, when $m \approx n$ (for example, in square matrices), it is possible that the number of non-maximal elimination sequences is larger than that of maximal ones. Thus our model may not be a good model. An open problem is therefore to find a new model for row elimination that will take into account both non-maximal and maximal elimination sequences, and to have a better understanding of the effect of non-maximal elimination sequences.

(3) A problem that we have considered briefly in Chapter 6 is that of identifying dense rows, which is usually done by examining the number of nonzeros in each row. The open question is whether it is worthwhile to use a more sophisticated way of identifying dense rows. If it

is, then a different definition of dense rows is required and a more robust method of identifying them is needed.

(4) We have derived updating algorithms for solving many classes of linear systems using the QR-decomposition of a sparse portion of A and the withheld dense rows. Updating algorithms for rank-deficient underdetermined systems with dense columns remain to be developed. Another problem is to compare various updating algorithms for solving a particular class of systems (in particular, linear least squares problems) in terms of storage requirement, execution time and numerical stability.

(5) In this thesis we have assumed that the ordering algorithms can be carried out in-core. However, there are large problems in which orderings cannot be performed in main storage, and secondary storage must be used. Hence an important problem to be solved is to design algorithms that will produce, for example, a minimum degree ordering or a width-1 nested dissection ordering, using secondary storage.

(6) Another problem is to find out the difficulty of the ordering problems. We know that finding an optimal column ordering for A is an NP-complete problem. Is the problem of finding an optimal row ordering for A, given a column ordering, also NP-complete?

LIST OF REFERENCES

[1] R.J. ALLWOOD, "Matrix methods of structural analysis", in *Large sparse sets of linear equations*, ed. J.K. Reid, Academic Press, London (1971), pp. 17-24.

[2] I. ARANY, W.F. SMYTH, AND L. SZODA, "An improved method for reducing the bandwidth of sparse symmetric matrices", in *Information Processing 71*, , Proc. of IFIP Congress, North-Holland, Amsterdam (1972),

[3] J.H. ARGYRIS AND O.E. BRONLUND, "The natural factor formulation of the stiffness matrix displacement method", *Comput. Meth. Appl. Mech. Engrg.*, **5** (1975), pp. 97-119.

[4] J.H. ARGYRIS, T.L. JOHNSEN, AND H.P. MLEJNEK, "On the natural factor in nonlinear analysis", *Comput. Meth. Appl. Mech. Engrg.*, **15** (1978), pp. 389-406.

[5] J.P. BATY AND K.L. STEWART, "Dissection of structures", *J. Struc. Div. ASCE*, **5** (1967), pp. 217-232.

[6] J.P. BATY AND K.L. STEWART, "Organization of network equations using dissection theory", in *Large sparse sets of linear equations*, ed. J.K. Reid, Academic Press, London (1971), pp. 169-190.

[7] C. BERGE, *The theory of graphs and its applications*, John Wiley & Sons Inc., New York (1962).

[8] G. BIRKHOFF AND J.A. GEORGE, "Elimination by nested dissection", in *Complexity of sequential and parallel numerical algorithms*, ed. J.F. Traub, Academic Press, New York (1973), pp. 21-269.

[9] A. BJORCK, "Methods for sparse linear least squares problems", in *Sparse matrix computations*, ed. J.R. Bunch and D.J. Rose, Academic Press, New York (1976), pp. 177-199.

[10] A. BJORCK, "A general updating algorithm for constrained linear least squares problems", Tech. Report LiTH-MAT-R-81-18, Dept. of Math., University of Linkoping, Sweden (1981).

[11] Y.T. CHEN AND R.P. TEWARSON, "On the fill-in when sparse vectors are orthonormalized", *Computing*, **9** (1972), pp. 53-56.

[12] R.E. CLINE AND R.J. PLEMMONS, "l_2-solutions to underdetermined linear systems", *SIAM Review*, **18** (1976), pp. 92-106.

[13] J.W. DANIEL, W.B. GRAGG, L. KAUFMAN, AND G.W. STEWART, "Reorthogonalization and stable algorithms for updating the Gram-Schmidt *QR* factorization", *Math. Comp.*, **30** (1976), pp. 772-795.

[14] J.J. DONGARRA, C.B. MOLER, J.R. BUNCH, AND G.W. STEWART, *LINPACK users' guide*, SIAM, Philadelphia (1980).

[15] I.S. DUFF, "Pivot selection and row ordering in Givens reduction on sparse matrices", *Computing*, **13** (1974), pp. 239-248.

[16] I.S. DUFF, A.M. ERISMAN, AND J.K. REID, "On George's nested dissection method", *SIAM J. Numer. Anal.*, **13** (1976), pp. 686-695.

[17] 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. Appl.*, **17** (1976), pp. 267-280.

[18] R.W. FAREBROTHER, "An historical note on the least squares updating formulas", Report, University of Manchester (1978).

[19] K.O. GEDDES, G.H. GONNET, AND B.W. CHAR, "MAPLE user's manual, second edition", Research report CS-82-40, Department of Computer Science, University of Waterloo, Waterloo, Ontario (1982).

[20] W.M. GENTLEMAN, "Regression problems and the *QR* decomposition", *Bull. Inst. Math. Appl.*, **10** (1974), pp. 195-197.

[21] W.M. GENTLEMAN, "Error analysis of *QR* decompositions by Givens transformations", *Linear Algebra and its Appl.* , **10** (1975), pp. 189-197.

[22] W.M. GENTLEMAN, "Row elimination for solving sparse linear systems and least squares problems", in *Proc. 1975 Dundee Conference on Numerical Analysis*, ed. G.A. Watson, Lecture Notes in Mathematics (506), Springer-Verlag (1975), pp. 122-133.

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

[24] J.A. GEORGE, "An automatic one-way dissection algorithm for irregular finite element problems", *SIAM J. Numer. Anal.*, **17** (1980), pp. 740-751.

[25] J.A. GEORGE AND M.T. HEATH, "Solution of sparse linear least squares problems using Givens rotations", *Linear Algebra and its Appl.*, **34** (1980), pp. 69-83.

[26] J.A. GEORGE, M.T. HEATH, AND E.G.Y. NG, "A comparison of some methods for solving sparse linear least squares problems", *SIAM J. Sci. Stat. Comput.*, (1983), (To appear.)

[27] J.A. GEORGE, M.T. HEATH, AND R.J. PLEMMONS, "Solution of large-scale sparse least squares problems using auxiliary storage", *SIAM J. Sci. Stat. Comput.*, **2** (1981), pp. 416-429.

[28] J.A. GEORGE AND J.W.H. LIU, "An automatic nested dissection algorithm for irregular finite element problems", *SIAM J. Numer. Anal.*, **15** (1978), pp. 1053-1069.

[29] J.A. GEORGE AND J.W.H. LIU, "Algorithms for matrix partitioning and the numerical solution of finite element systems", *SIAM J. Numer. Anal.*, **15** (1978), pp. 297-327.

[30] J.A. GEORGE AND J.W.H. LIU, "The design of a user interface for a sparse matrix package", *ACM Trans. on Math. Software*, **5** (1979), pp. 134-162.

[31] J.A. GEORGE AND J.W.H. LIU, "A quotient graph model for symmetric factorization", in *Sparse matrix proceedings 1978*, ed. I.S. Duff and G.W. Stewart, SIAM, Philadelphia (1979), pp. 154-175.

[32] J.A. GEORGE AND J.W.H. LIU, "An implementation of a pseudo-peripheral node finder", *ACM Trans. on Math. Software*, **5** (1979), pp. 286-295.

[33] J.A. GEORGE AND J.W.H. LIU, "A minimal storage implementation of the minimum degree algorithm", *SIAM J. Numer. Anal.*, **17** (1980), pp. 282-299.

[34] J.A. GEORGE AND J.W.H. LIU, "A fast implementation of the minimum degree algorithm using quotient graphs", *ACM Trans. on Math. Software*, **6** (1980), pp. 337-358.

[35] J.A. GEORGE AND J.W.H. LIU, "An optimal algorithm for symbolic factorization of symmetric matrices", *SIAM J. Comput.*, **9** (1980), pp. 583-593.

[36] J.A. GEORGE AND J.W.H. LIU, *Computer solution of large sparse positive definite systems*, Prentice-Hall Inc., Englewood Cliffs, N.J. (1981).

[37] J.A. GEORGE, J.W.H. LIU, AND E.G.Y. NG, "User's guide for SPARSPAK: Waterloo sparse linear equations package", Research report CS-78-30 (revised), Department of Computer Science, University of Waterloo (1980).

[38] J.A. GEORGE AND E.G.Y. NG, "On row and column orderings for sparse least squares problems", *SIAM J. Numer. Anal.*, **20** (1983), pp. 326-344.

[39] N.E. GIBBS, W.G. POOLE, JR., AND P.K. STOCKMEYER, "An algorithm for reducing the bandwidth and profile of a sparse matrix", *SIAM J. Numer. Anal.*, **13** (1976), pp. 236-250.

[40] J.R. GILBERT, "Graph separator theorems and sparse Gaussian elimination", Report No. STAN-CS-80-833, Dept. of Computer Science, Stanford University (1980).

[41] P.E. GILL AND W. MURRAY, "A numerically stable form of the simplex algorithm", *Linear Algebra and its Appl.*, **7** (1973), pp. 99-138.

[42] P.E. GILL AND W. MURRAY, "The orthogonal factorization of a large sparse matrix", in *Sparse matrix computations*, ed. J.R. Bunch and D.J. Rose, Academic Press, New York (1976), pp. 201-212.

[43] J.W. GIVENS, "Numerical computation of the characteristic values of a real symmetric matrix", Oak Ridge National Laboratory Report ORNL-1574 (1954).

[44] G.H. GOLUB, "Numerical methods for solving linear least squares problems", *Numer. Math.*, **7** (1965), pp. 206-216.

[45] G.H. GOLUB AND W. KAHAN, "Calculating the singular values and pseudo-inverse of a matrix", *SIAM J. Numer. Anal.*, **2** (1965), pp. 202-224.

[46] G.H. GOLUB AND R.J. PLEMMONS, "Large-scale geodetic least squares adjustment by dissection and orthogonal decomposition", *Linear Algebra and its Appl.*, **34** (1980), pp. 3-27.

[47] G.H. GOLUB AND C. REINSCH, "Singular value decomposition and least squares solutions", *Numer. Math.*, **14** (1970), pp. 403-420.

[48] M.C. GOLUMBIC, *Algorithmic graph theory and perfect graphs*, Academic Press, New York (1980).

[49] F.G. GUSTAVSON, "Some basic techniques for solving sparse systems of equations", in *Sparse matrices and their applications*, ed. D.J. Rose and R.A. Willoughby, Plenum Press, New York (1972), pp. 41-52.

[50] M.T. HEATH, "Some extensions of an algorithm for sparse linear least squares problems", *SIAM J. Sci. Stat. Comput.*, **3** (1982), pp. 223-237.

[51] H.V. HENDERSON AND S.R. SEARLE, "On deriving the inverses of a sum of matrices", *SIAM Review*, **23** (1981), pp. 53-60.

[52] A.J. HOFFMAN, M.S. MARTIN, AND D.J. ROSE, "Complexity bounds for regular finite difference and finite element grids", *SIAM J. Numer. Anal.*, **10** (1973), pp. 364-369.

[53] L. KAUFMAN, "Application of dense Householder transformations to a sparse matrix", *ACM Trans. on Math. Software*, **5** (1979), pp. 442-450.

[54] G.B. KOLATA, "Geodesy: dealing with an enormous computer task", *Science*, **200** (1978), pp. 421-422,466.

[55] C.L. LAWSON AND R.J. HANSON, *Solving least squares problems*, Prentice-Hall Inc., Englewood Cliffs, N.J. (1974).

[56] R.J. LIPTON, D.J. ROSE, AND R.E. TARJAN, "Generalized nested dissection", *SIAM J. Numer. Anal.*, **16** (1979), pp. 346-358.

[57] R.J. LIPTON AND R.E. TARJAN, "A separator theorem for planar graphs", *SIAM J. Appl. Math.*, **36** (1979), pp. 177-199.

[58] J.W.H. LIU, "The solution of mesh equations on a parallel computer", Research Report CS-74-19, Dept. of Computer Science, University of Waterloo (1974).

[59] J.K. PACHL, "Finding pseudoperipheral nodes in graphs", Research report CS-82-20, Department of Computer Science, University of Waterloo, Waterloo (1982).

[60] C.C. PAIGE, "An error analysis of a method for solving matrix equations", *Math. Comp.*, **27** (1973), pp. 355-359.

[61] S.V. PARTER, "The use of linear graphs in Gaussian elimination", *SIAM Review*, **3** (1961), pp. 364-369.

[62] R.L. PLACKETT, "Some theorems in least squares", *Biometrika*, **37** (1950), pp. 149-157.

[63] U.W. POOCH AND A. NEIDER, "A survey of indexing techniques for sparse matrices", *ACM Computing Survey*, **5** (1973), pp. 109-133.

[64] D.J. ROSE, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", in *Graph theory and computing*, ed. R.C. Read, Academic Press (1972), pp. 183-217.

[65] D.J. ROSE, R.E. TARJAN, AND G.S. LUEKER, "Algorithmic aspects of vertex elimination on graphs", *SIAM J. Comput.*, **5** (1976), pp. 266-283.

[66] D.J. ROSE AND G.F. WHITTEN, "A recursive analysis of dissection strategies", in *Sparse matrix computations*, ed. J.R. Bunch and D.J. Rose, Academic Press (1976), pp. 59-84.

[67] M.A. SAUNDERS, "Large-scale linear programming using the Choleksy factorization", Computer Science Department Report No. CS 252, Stanford University, Stanford, Calif. (1972).

[68] A.H. SHERMAN, "On the efficient solution of sparse systems of linear and nonlinear equations", Research Report #46, Dept. of Computer Science, Yale University (1975).

[69] J. SHERMAN AND W.J. MORRISON, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix", *Annals of Mathematical Statistics*, **20** (1949), p. 621.

[70] W.F. SMYTH AND W.M.L. BENZI, "An algorithm for finding the diameter of a graph", *IFIP Congress 74*, (1974), North-Holland Publ. Co., pp. 500-503.

[71] G.W. STEWART, *Introduction to matrix computations*, Academic Press, New York (1973).

[72] R.E. TARJAN, "Graph theory and Gaussian elimination", in *Sparse matrix computations*, ed. J.R. Bunch and D.J. Rose, Academic Press, New York (1976), pp. 3-22.

[73] R.P. TEWARSON, "On the orthonormalization of sparse vectors", *Computing*, **3** (1968), pp. 268-279.

[74] J.H. WILKINSON, *The algebraic eigenvalue problem*, Oxford University Press, Oxford (1965).

[75] M. WOODBURY, "Inverting modified matrices", Memorandum Report 42, Statistical Research Group, Princeton University (1950).

[76] M. YANNAKKIS, "Computing the minimum fill-in is NP-complete", *SIAM J. Alg. Disc. Meth.*, **2** (1981), pp. 77-79.

[77] Z. ZLATEV, "Comparison of two pivotal strategies in sparse plane rotations", *Comp. and Maths. with Appls.*, **8** (1982), pp. 119-135.