ON REDUCING THE PROFILE OF SPARSE SYMMETRIC MATRICES

Joseph Wai-Hung Liu
Department of Computer Science
University of Waterloo
Waterloo, Ontario

CS-76-07

February 1976

ON REDUCING THE PROFILE OF SPARSE SYMMETRIC MATRICES

by

JOSEPH WAI-HUNG LIU

A THESIS SUBMITTED

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

AT THE

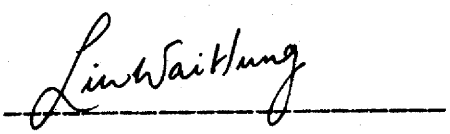
UNIVERSITY OF WATERLOO

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF MATHEMATICS.

ONTARIO, CANADA.

NOVEMBER, 1975.

I hereby declare that I am the sole author of this thesis. I authorize the University of Waterloo to lend it to other institutions or individuals for the purpose of scholarly research.

Signature 

Acknowledgement

I wish to express my heartfelt gratitude and appreciation to my thesis supervisor, Professor Alan George, for his unfailing support and constant encouragement during the years of my graduate study. He has been most understanding and considerate.

I also like to thank the reading committee, Professor M. Malcolm, Professor I. Munro, Professor W. Poole, Jr., and Professor R. Read for their valuable suggestions and comments.

Financial support from the National Research Council of Canada and an Ontario Graduate Fellowship is gratefully acknowledged.

To My Parents

& Josephine

Abstract

This thesis examines the envelope or profile method for solving symmetric, positive definite, sparse linear systems. This method is a variation of the popular band method, where only zeros outside a particular region in the matrix are exploited. It is known to be an effective scheme with simple data structures and convenient programming.

In our study, a graph-theoretic approach is used to investigate the reordering problem for profile reduction of various classes of graphs. The minimum and minimal envelope orderings are defined. An $O(N \log_2 N)$ algorithm is presented, which always generates a minimal envelope ordering for trees of N nodes.

Profile ordering algorithms for general graphs are largely heuristic. Existing effective algorithms are analysed, compared and tested. Generally, the reverse Cuthill-McKee algorithm is found to be the most consistently effective method.

Finite element graphs arise in the application of finite element methods for the solution of partial differential equations. They have special structures that can be exploited in profile minimization. The idea of element annihilation, which originates with the work of Irons, is used to develop a new ordering scheme. It is shown to be a practical alternative for certain classes of finite element graphs.

Table of Contents

Chapter 1 General Introduction

1.1 Statement of Problem.

In this thesis, we are concerned with the direct numerical solution of an N by N linear algebraic system:

$$A x = b, \qquad (1.1)$$

where the matrix A is sparse, symmetric and positive definite. Such systems arise frequently in applications, notably in the solution of elliptic boundary value problems by finite difference and finite element methods ([Varga62], [Zienkiewicz70]).

In general, an N by N matrix A is sparse if it has relatively few nonzero components. It is customary to measure the density of the matrix A by the quantity ([Pooch73], [Wang73], [Birkhoff73]):

$$d(A) = \gamma(A)/N^2,$$

where $\gamma(A)$ is the number of nonzeros in A. Typically, the matrix density $d(A) \leq O(N^{-1})$. The sparsity of A can then be measured by $1 - d(A)$.

Most direct solution methods for a positive definite symmetric system are based on the Cholesky factorization $A = L L^T$, where L is a lower triangular matrix, or on the related factorization $A = \bar{L}\bar{D}\bar{L}^T$, where $\bar{L}$ is unit lower triangular and $\bar{D}$ is a positive diagonal matrix. For symmetric positive definite matrices, such decomposition algorithms

are known to be numerically stable [Wilkinson65], so that no row or column interchange is necessary. The difference between the effectiveness of the two factorizations is insignificant. "There is marginally more work with the Cholesky decomposition but it is marginally simpler from the point of view of the organization involved." [Martin65]. For our study, we shall use the Cholesky LL^T, but our analysis also applies to the variant form $\bar{L}\bar{D}\bar{L}^T$.

To solve the system (1.1), the complete process consists of two major steps: the symmetric factorization (or decomposition) of the matrix A and the back substitution for the solution vector x. With the matrix A decomposed into the product:

$$A = L\,L^T, \tag{1.2}$$

the solution x can be obtained by performing the forward and backward substitution:

$$L\,y = b, \quad L^T x = y. \tag{1.3}$$

In numerical linear algebra, this method of solution for dense linear systems is standard. Besides being numerically stable, the method is very economical in terms of the number of arithmetic operations. If N is the order of the dense system, the factorization requires

$$\tfrac{1}{6}N^3 + \tfrac{5}{4}N^2 - \tfrac{17}{2}N \quad \text{multiplicative operations}$$

and (1.4)

$$\tfrac{1}{6}N^3 + \tfrac{1}{4}N^2 - \tfrac{5}{2}N \quad \text{additions.}$$

The back-solving step (1.3) requires

$$N^2 \text{ multiplicative operations}$$

and (1.5)

$N^2 - N$ additions.

The set of procedures in ALGOL 60 by Martin, Peters and Wilkinson [Martin65] is tailored for solving systems (1.1) with dense coefficient matrices. In their implementation, symmetry in the matrix is exploited by storing only its lower triangle row by row in a linear array. The storage requirement is reduced to

$\frac{1}{2} N(N+1)$ locations. (1.6)

When the coefficient matrix is large and sparse, the number of arithmetic operations in (1.4) and (1.5), and the amount of storage in (1.6) may be prohibitively large. These requirements can be drastically reduced if sufficient care is taken to avoid storing and operating on some or all zero components in the matrix. By exploiting the sparseness structure, it has become quite common to solve linear systems with several thousand unknowns by direct methods.

Now, it is well known that when symmetric Gaussian elimination is applied to a sparse matrix A, it usually suffers some fill; that is, the lower triangular factor L has nonzero components in positions which are zero in the original matrix. This phenomenon of fill-in, which is usually ignored in solving dense systems, plays an important role in sparse elimination. It not only affects the computation and storage requirements, but also the data structure and hence the coding of the algorithm.

In general, we do not know a priori the exact locations and the exact number of fills unless a simulation of the factorization process (often called symbolic factorization) is performed beforehand. Thus, a flexible packed storage scheme for the sparse matrix A is essential for the successful implementation of the algorithm. The storage method should be able to handle fill-in terms with relative ease. Furthermore, the storage and indexing overhead should be small so that it does not offset the reduction in arithmetic and primary storage.

Reordering is another important issue in the study of sparse techniques. For a fixed compact storage scheme, the ordering of the equations in a given sparse matrix problem can greatly affect the amount of arithmetic and storage required for its direct solution. More specifically, if P is a permutation matrix, the matrices A and PAP^T may require different amount of computation for their respective factorizations. Moreover, their triangular factors generally have different numbers of fills. Since the permuted linear system

$$(PAP^T)(Px) = Pb, \qquad (1.7)$$

remains sparse, symmetric and positive definite, PAP^T always possesses a triangular factorization which can be determined in a numerically stable manner. Thus, it may be more advantageous in terms of computational and storage cost to solve (1.7) instead of (1.1). It should be noted

that the best choice of P depends greatly on the storage method used.

The significance of reordering before solving linear systems has been well recognised. Most solvers include it as a major step. Figure 1.1 is a flow-chart for a general purpose solution package for sparse systems (1.1).

In this thesis, we give a detailed analysis of the envelope method, a sparse storage method that exploits zeros outside a particular region in the matrix A. This region is known to contain the set of matrix positions at which fill occurs during factorization. Our main concern is on the reordering problem; we investigate ways to permute the rows and columns of a matrix so that the envelope method can be applied most effectively.

The next section is a brief summary of the currently used direct sparse techniques. It is intended to bring out the significance of the envelope method and its similarities to and differences from the other methods.

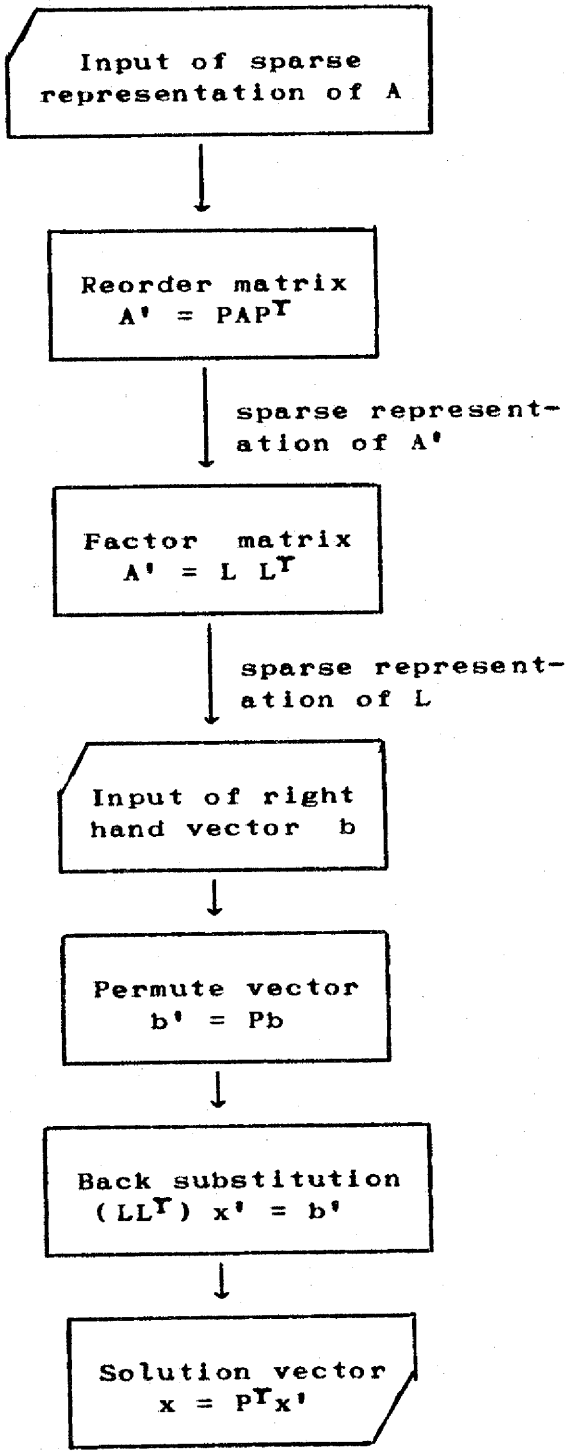


Figure 1.1: Flowchart for general solver

1.2 Survey of Direct Sparse Techniques

Sparse matrix technology is a fast expanding field with a wide spectrum of applications. The survey articles [Christian73], [George73b] and [Willoughby72] review the state-of-the-art of this technology. In this section, we present a rudimentary survey on direct sparse methods as related to the envelope scheme.

Direct sparse methods based on triangular factorization fall into three fundamental classes: general sparse methods, partitioning/block methods, and band or envelope methods. The basic difference between these methods lies in the care taken to avoid the operations and storage for zero components. But in practice, this leads to differences in many aspects: data representation, data management, adaptation to memory-hierarchy environment, reordering algorithms and implementation complexity.

General sparse methods exploit all the zeros in the matrix A and its triangular factor L. To this effect, a packed matrix storage scheme is required so that only nonzero entries are stored. Besides primary storage for the matrix terms, additional indexing information has to be kept to identify the terms and to facilitate their addressing. In [Churchill71], [Jennings68], [Larcombe71] and many others, linked lists are used, which prove to be very flexible in accomodating fill-in terms during the factorization. A discussion of the basic techniques

involved is given in [Gustavson72]. Bit map is another common form of representation, especially in symbolic factorization [Gustavson70]. The arc-graph structure is a conceptually different representation, recently developed by Rheinboldt and Mesztenyi [Rheinboldt73]. Its potential advantage is that the scheme is independent of row/column permutation.

In [Gustavson70], Gustavson, Liniger, and Willoughby have developed a novel approach, which is quite different from conventional implementations. They have a symbolic preprocessing code, which generates a string of nonlooping machine instructions from the sparse structure of a given matrix. The generated machine code, when invoked, performs the actual numerical factorization and solution. So, repeated solution of a set of linear equations having a fixed structure but variable coefficients can be computed very efficiently. The implementations of [Chang69], [Lee69] and [Symposium75,Millar] utilize the same idea.

For the class of general sparse methods, the number of fills plays a crucial role; it affects both the storage and computation requirements. A graph-theoretic study of the fill phenomenon is given in [Parter61] and [Rose72a]. For judiciously chosen matrix permutations, the number of fills can be significantly reduced. The minimum degree and minimum deficiency ordering algorithms have been proposed; for descriptions of the algorithms, see [Rose72a] or [Ogbuobiri70]. Indeed, the minimum degree algorithm is the

most widely used scheme for reducing fills on general graphs.

In minimizing fills, George's nested dissection ordering is important. For linear systems associated with an n by n regular grid (n^2 equations), it has been shown that this ordering creates at most $O(n^2 \log_2 n)$ number of fills and requires $O(n^3)$ operations for factorization [George73a], which are optimal in the order of magnitude sense [Hoffman73]. This ordering strategy has a profound impact on the current research in general sparse methods [Symposium75].

In general, numbering matrices to achieve near minimum fill often leads to triangular factors which have their nonzero components scattered throughout the matrix. Data management problems arise when such systems are to be solved on a two-level store. As Tinney remarks [Tinney69], the class of general sparse techniques "does not lend itself to overlay".

In contrast, one of the motivations for partition/block methods is to facilitate the use of peripheral storage when the matrix problem is too large for the main memory. The problem is segmented so that the matrix subproblems can be solved successively in core. There are other good reasons for doing partitioning, such as better project management, reduced storage, and reduced operations [George73b].

Partition methods exploit sparsity by storing and processing the matrix by blocks. Zero submatrices are never stored, but non-null blocks are usually treated as full. Although some zero entries are stored and processed, block methods have the advantages of better data management and reduced bookkeeping overhead than general methods.

The hypermatrix method [Fuchs72] is a storage management scheme for block methods. Besides storing the non-zero blocks, the scheme maintains an address matrix to identify the locations of the blocks in the storage pool. A zero entry in the address matrix denotes a zero submatrix. Another storage scheme for partitioned matrices is described in [George75a], and it is specially tailored for the nested dissection ordering on grid problems.

In [George74], George gives an interesting discussion on the different ways block factorization can be carried out. In this connection, Bunch and Rose [Bunch74] observe that operations and storage can sometimes be saved by having the off-diagonal blocks of the triangular factor L only implicitly. In effect, this implicit storage scheme stores the diagonal blocks of L and the off-diagonal blocks of the original matrix A. When required, the off-diagonal submatrices of L are generated through their definitions. In [George75c], George and Liu extend this idea to "tree" partitionings, and demonstrate substantial storage saving in practical problems.

In many applications, for example in structural analysis, partitioning is usually obtained as imposed by the physical nature of the problem. Automatic partitioning is rarely discussed in the literature. Notable exceptions are the one-way dissection [George75a] and the "quotient tree" partitioning scheme [George75c]. The design of automatic partitioning schemes is a promising area for research.

Unlike the previous methods, band and band-like methods take advantage of the zero structure of the matrix by storing only those entries within particular regions of the matrix. These regions are chosen so that all the nonzero entries in the matrix and the fills due to factorization lie within them. The methods generally store and operate on more zeros in exchange for much simpler data management and programming.

The band method is well-known and enjoys great popularity in solving partial differential equations. It has simple data structures and is extremely well adapted to overlay procedures. The diagonal storage scheme for banded matrices [Martin65] is standard.

The crucial parameter in band schemes is the bandwidth (see section 2.1). It becomes important to have algorithms for generating orderings with narrow bandwidths. Algorithms suggested include those of [Alway65], [Arany71], [Cheng73a], [Collins73], [Cuthill69], [Gibbs74a],

[Rosen68], and [Wang73]. A comparative study of them can be found in [Cuthill72] and [Gibbs74b]. The recent algorithm by Gibbs, Poole and Stockmeyer is the currently best bandwidth reduction scheme.

The envelope method takes advantage of variations in the row bandwidths of the matrices. In the literature, it also goes under the names of the profile, frontal, and wavefront schemes ([George71], [Irons70], [Jennings66], [Melosh69]). The method requires no more storage and computation† than the band method, and in most cases significantly less. For problems of practical interest, over 50% of storage saving has been reported [George71]. Yet, the method retains the advantages of band schemes: simple and compact data structure, efficient data management and convenient coding. It is a method of practical importance.

Though the method arises primarily as a refinement to the band scheme, the reordering strategies are based on different criterion functions. An analysis of the reordering problem for profile reduction is the main theme of the present thesis.

---

† Many implementations of the band method perform routine checking of zero multipliers, so that for the same ordering, they require the same amount of computation as the envelope method. [Wilson74]

1.3 Outline of Thesis

The purpose of this thesis is to study the envelope or profile method for sparse, positive definite, symmetric systems. As indicated earlier, the method enjoys simple data management and convenient programming, while still retains practical efficiency. In chapter 2, we examine the implementational aspects of the method. An analysis of the storage and computation requirement is presented.

Chapter 3 is devoted to a study of the profile method using a graph-theoretic approach. We introduce envelope structures for graphs, so that the minimum envelope and minimal envelope problems can be formulated. Minimum envelope orderings are ideal to have, while minimal orderings are important from a graph-theoretic point of view. Some properties of these and their related orderings are established in sections 3.3-3.5.

Trees form a class of simple graphs. In chapter 4, we consider the reordering problem on tree structures for small profiles. The familiar postorder traversal scheme is used recursively to yield an $O(N \log_2 N)$ algorithm which always generates a minimal envelope ordering for trees of N nodes. Some experiments with the algorithm are reported in section 4.5. Readers not interested in the theoretical problem of determining minimal orderings can skip sections 3.3-3.4 and chapter 4.

The objective of chapter 5 is to examine the reduction problem for general graphs (and hence general symmetric matrices). Existing profile ordering algorithms for general graphs are heuristic; their performances are data-dependent. Several popular profile reduction schemes are analysed and compared. Included in the study are the Cuthill-McKee, reverse Cuthill-McKee, King and Levy algorithms. We use a graph-theoretic approach to show that the reverse Cuthill-McKee algorithm can never be inferior to the Cuthill-McKee algorithm in envelope storage and computational requirements. A matrix proof of this result has been given by Liu and Sherman [Liu75].

No such theoretical comparisons can be established for the other algorithms. Instead, the comparisons are based on experimental testings. The recent algorithm by Gibbs, Poole and Stockmeyer is also included in the experiments. Generally speaking, the reverse Cuthill-McKee algorithm performs most consistently and quite effectively.

Chapter 6 discusses the profile problem for finite element graphs, a practical class of graphs arising from the application of the finite element method to two dimensional problems. We observe that the geometry of the graph problem can be exploited for this purpose. The idea of element annihilation, which originated with the work of [Irons70], is shown to be an alternative approach for

studying profile minimization. Based on this, a new profile scheme is proposed. Through extensive testing, it is shown to be a viable alternative for certain classes of finite element graphs.

In the thesis, algorithms are described in an ALGOL-like language. FORTRAN implementations of them can be found in the appendix.

1.4 Review of Symmetric Factorization

In this section, we review the process of symmetric factorization and we make some observations on the solution of triangular systems.

The step by step triangular factorization of a given N by N matrix A into LL^T may be described by the following outer-product formulation [Westlake69].

$$A = A_0 = B_0 = \begin{bmatrix} d_1 & v_1^T \\ v_1 & \bar{B}_1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{d_1} & 0 \\ \dfrac{v_1}{\sqrt{d_1}} & I_{N-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \bar{B}_1 - \dfrac{v_1 v_1^T}{d_1} \end{bmatrix} \begin{bmatrix} \sqrt{d_1} & v_1^T/\sqrt{d_1} \\ 0 & I_{N-1} \end{bmatrix}$$

$$= L_1 \begin{bmatrix} 1 & 0 \\ 0 & B_1 \end{bmatrix} L_1^T$$

$$= L_1 A_1 L_1^T,$$

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & B_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \\ d_2 & v_2^T & \\ 0 & v_2 & \bar{B}_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \\ \sqrt{d_2} & 0 & \\ 0 & \frac{v_2}{\sqrt{d_2}} & I_{N-2} \end{bmatrix} \begin{bmatrix} 1 & 0 & \\ 1 & 0 & \\ 0 & 0 & \bar{B}_2 - \frac{v_2 v_2^T}{d_2} \end{bmatrix} \begin{bmatrix} 1 & 0 & \\ \sqrt{d_2} & v_2^T/\sqrt{d_2} & \\ 0 & 0 & I_{N-2} \end{bmatrix}$$

$$= L_2 \; A_2 \; L_2^T,$$

$$\cdot \quad \cdot \quad \cdot \qquad (1.8)$$

$$A_{N-1} = L_N \; I_N \; L_N^T.$$

Here, for $1 \le i \le N$, d_i is a positive scalar, v_i is a vector of length $N-i$, and B_i is an $N-i$ by $N-i$ positive definite symmetric matrix.

After N steps of the algorithm, we have $A = LL^T$, where $L = L_1 L_2 \cdots L_N$. Note that the triangular factor L is also given by:

$$L = \left(\sum_{i=1}^{N} L_i \right) - (N-1) \; I_N, \qquad (1.9)$$

which implies that the i-th column of L is precisely the i-th column of L_i.

Each factorization step involves the modification of the submatrix $\bar{B}_i$ by the outer product $(v_i v_i^T)/d_i$ to give B_i, which becomes the submatrix remaining to be factored. As a result, the submatrix B_i may have nonzeros in locations which are zero in $\bar{B}_i$. Thus, the matrix A generally fills-in, and with the assumption that exact numerical cancellation does not occur during the factorization, the matrix sum $L+L^T$ is usually fuller than A.

Besides giving a better understanding of why and how a matrix fills, the outer-product formulation provides a direct way of determining the exact arithmetic cost. Let $\gamma(\bullet)$ denote the number of nonzero components in $\bullet$, where $\bullet$ may be a matrix or a vector. The following lemma is by Rose [Rose72a].

Lemma 1.1 Provided we avoid operating on and storing all zeros, the number of multiplicative operations required to perform the Cholesky factorization LL^T of A is given by

$$\theta(A) = \tfrac{1}{2} \sum_{i=1}^{N-1} \gamma(v_i)\,\{\gamma(v_i)+3\}, \qquad (1.10)$$

and the number of nonzero components in L is:

$$\gamma(L) = \sum_{i=1}^{N-1} \gamma(v_i) + N. \qquad (1.11)$$

□

If we use L_{*i} to denote the i-th column of L, then we have $\gamma(L_{*i}) = \gamma(v_i)+1$. Thus, (1.10) can be rewritten as:

$$\theta(A) = \frac{1}{2} \sum_{i=1}^{N-1} \{\gamma(L_{*i})-1\} \{\gamma(L_{*i})+2\}.$$

An alternative formulation of the symmetric factorization process uses <u>inner-products</u>. It is sometimes known as the <u>bordering</u> method. Suppose the symmetric positive definite N by N matrix A is partitioned as:

$$A = \begin{bmatrix} M & u \\ u^T & s \end{bmatrix}, \qquad (1.12)$$

where the symmetric factorization $L_M L_M^T$ of the N-1 by N-1 matrix M has already been obtained. Here, u is a vector of length N-1 and s is a scalar. Then, the factorization of A is given by:

$$A = \begin{bmatrix} L_M & 0 \\ l^T & t \end{bmatrix} \begin{bmatrix} L_M^T & l \\ 0 & t \end{bmatrix}, \qquad (1.13)$$

where $L_M\, l = u,$ (1.14)

and $t^2 = s - l^T l.$ (1.15)

Note that, in practice, the factorization $L_M L_M^T$ of the principal submatrix M is also obtained by the bordering technique. Thus, the entire process of factoring A into triangular factors consists of N bordering steps, each

involving the solution of a lower triangular system (1.14) and the computation of an inner product (1.15).

The outer-product and inner-product approaches require the same amount of arithmetic; they only differ in the order of operations. The former can be considered as factorization by columns and the latter by rows. Figure 1.2 shows the two formulations in a pictorial manner.

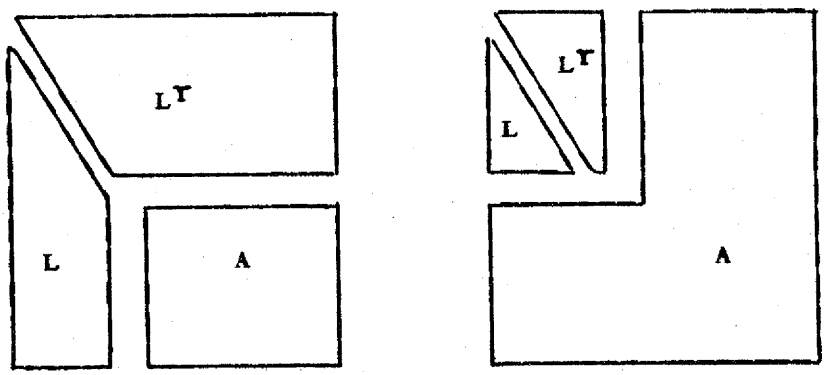


Figure 1.2: Two Formulations of Factorization

Conceptually, the outer-product formulation gives better insight into the elimination process. It is often used in symbolic factorizations (or factorizations that exploit all zeros). But in practice, the bordering method is usually preferred ([Martin65], [Tinney69]), primarily because it is more adapted to row-wise storage schemes.

Solutions to triangular systems are required in the back substitution phase (1.3) and in the bordering formulation (1.14) of the factorization phase. We shall establish some observations on solving such systems. Consider the N by N linear system:

$$T\,x = b, \qquad (1.16)$$

where T is triangular and nonsingular. For definiteness, we assume that T is lower triangular; similar results can be established when T is upper triangular. The following results are by George [George74].

Lemma 1.2: With the no cancellation assumption, if the component b_i is nonzero, so is x_i.

Proof: Let T_{ij} be the (i,j)-th component of T. Since T is nonsingular, $T_{ii} \neq 0$ for $1 \leq i \leq N$. The result then follows from the no cancellation assumption and the relation: i=2,..,N

$$x_i = \{b_i - \sum_{j=1}^{i-1} T_{ij}\, x_j\}/T_{ii}. \qquad \square$$

Lemma 1.3: If the sparsity of the right hand vector b is not exploited, the number of multiplicative operations required to solve (1.16) is given by $\gamma(T)$, the number of nonzero components in T.

Proof: It follows from the result of lemma 1.2 and the observation that the number of multiplicative operations is equal to the number required to multiply T by the vector x. □

The converse of lemma 1.2 does not hold in general; we only have

Lemma 1.4: Let x be the solution to Tx=b. If $b_i=0$ for $1\le i\le k$, then $x_i=0$ for $1\le i\le k$. □

The above lemma, though trivial, is useful in showing that fill is confined to some particular region of a matrix (section 2.2).

Chapter 2 On the Envelope Method

In this chapter, we will give a detailed study of the envelope method in the direct solution of sparse symmetric, positive definite linear systems. The emphasis is on implementation aspects of the method. We consider a compact envelope storage scheme which is well suited for both the factorization and substitution processes. An analysis of the computational and storage requirements is also given.

2.1 Matrix Notations

In this section, we present some matrix notations and definitions that are relevant in the study of the envelope method. Let A be an N by N symmetric matrix with nonzero diagonal components. We denote the (i,j)-th entry of A by A_{ij}. For the i-th row of A, i=1,...,N, we let

$$f_i(A) = \min \{j: A_{ij} \neq 0\}, \tag{2.1}$$

$$\beta_i(A) = i - f_i(A). \tag{2.2}$$

The quantity $f_i(A)$ is the column subscript of the first nonzero component of the i-th row of A.

Following Cuthill and McKee [Cuthill69], we define the bandwidth of A by

$$\beta(A) = \max \{|i-j|: A_{ij} \neq 0\}. \tag{2.3}$$

Note that $\beta(A)$ can be expressed in terms of $\beta_i(A)$ as:

$$\beta(A) = \max \{\beta_i(A): 1 \leq i \leq N\},$$

so that $\beta_i(A)$ is called the i-th bandwidth of A. We can

then define the band of A as:

$$\text{Band}(A) = \{ \{i,j\}: 0 < i-j \leq \beta(A) \}, \qquad (2.4)$$

which is the region within $\beta(A)$ locations from the main diagonal. Unordered pairs $\{i,j\}$ are used in (2.4) instead of ordered (i,j), because the matrix A is symmetric.

Taking advantage of the variation in the band, we introduce the envelope structure of a matrix. The envelope of A is defined as:

$$\text{Env}(A) = \{ \{i,j\}: 0 < i-j \leq \beta_i(A) \}. \qquad (2.5)$$

In terms of the column subscripts $f(A)$, we note that

$$\text{Env}(A) = \{ \{i,j\}: f_i(A) \leq j < i \}. \qquad (2.6)$$

In [Segethova70], rows of the envelope are called pipes. The quantity $|\text{Env}(A)|$ [†] is usually referred to as the envelope size or the profile of A. It is clear that

$$|\text{Env}(A)| = \sum_{i=1}^{N} \beta_i(A), \qquad (2.7)$$

where $\beta_1(A)=0$.

Assume that the matrix A has the Cholesky factorization LL^T. It is well known that the lower triangular factor L may have nonzeros in locations that are zero in the original matrix A. (See section 1.2.) The set of such entries forms part of the primary storage of any sparse storage scheme for L. It is then natural to define the fill of A to be:

$$\text{Fill}(A) = \{ \{i,j\}: A_{ij}=0, (L+L^T)_{ij} \neq 0 \}. \qquad (2.8)$$

This is the set of locations where fill may occur.

---

† $|*|$ is used to denote the cardinality of the set $*$.

Consider the matrix example in figure 2.1. It has a bandwidth of 3 and an envelope size of 11.

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

i	$f_i(A)$	$\beta_i(A)$
1	1	0
2	1	1
3	3	0
4	1	3
5	3	2
6	3	3
7	5	2

$$\begin{bmatrix} x & x & & x & & & \\ x & x & & \blacksquare & & & \\ & & x & & x & x & \\ x & \blacksquare & & x & x & & \\ & & x & x & x & \blacksquare & x \\ & & x & & \blacksquare & x & x \\ & & & & x & x & x \end{bmatrix}$$

Fill(A)

$$\begin{bmatrix} x & x & & x & & & \\ x & x & & \blacksquare & & & \\ & & x & o & x & x & \\ x & \blacksquare & o & x & x & o & \\ & & x & x & x & \blacksquare & x \\ & & x & o & \blacksquare & x & x \\ & & & & x & x & x \end{bmatrix}$$

Env(A)

$$\begin{bmatrix} x & x & o & x & & & \\ x & x & o & \blacksquare & o & & \\ o & o & x & o & x & x & \\ x & \blacksquare & o & x & x & o & o \\ & o & x & x & x & \blacksquare & x \\ & & x & o & \blacksquare & x & x \\ & & & o & x & x & x \end{bmatrix}$$

Band(A)

Figure 2.1: A 7 by 7 symmetric matrix.

The band and envelope structures are important because zeros outside these regions can be exploited conveniently in the symmetric factorization of the matrix. In the next section, we shall see how the envelope structure can be exploited using an appropriate storage scheme. We now

relate structures (2.4), (2.5), and (2.8) together as follows.

Lemma 2.1: $Env(A) = Env(L+L^T)$, where $L\,L^T$ is the symmetric factorization of A.

Proof: We prove by induction on the dimension N. The result is clearly true when N is 1. Assume that the result holds for N-1 by N-1 matrices. Let A be an N by N symmetric matrix partitioned as:

$$A = \begin{bmatrix} M & u \\ u^T & s \end{bmatrix},$$

where s is a scalar, u a vector of length N-1, and M an N-1 by N-1 nonsingular matrix factored as $L_M L_M^T$. By the inductive assumption, we have $Env(M) = Env(L_M+L_M^T)$. The triangular factor L can be partitioned as:

$$L = \begin{bmatrix} L_M & 0 \\ l^T & t \end{bmatrix}$$

where t is a scalar, and l is a vector of length N-1. It is then sufficient to show that $f_N(A) = f_N(L+L^T)$.

From (1.14), the vectors u and l are related by:

$$L_M\, l = u.$$

But $u_i=0$ for $1 \le i < f_N(A)$ and the entry $u_{f_N(A)}$ is nonzero. By lemma 1.4, we have $l_i=0$ for $1 \le i < f_N(A)$ and by lemma 1.2, $l_{f_N(A)} \neq 0$. Hence $f_N(A) = f_N(L+L^T)$, so that

$$Env(A) = Env(L+L^T).$$ □

Theorem 2.1: $Fill(A) \subset Env(A) \subset Band(A)$.

Proof: The first inclusion follows from lemma 2.1 and the second from definition. □

2.2 Envelope Storage Scheme

The most commonly used storage scheme for the envelope method is the one proposed by Jennings [Jennings66]. For each row in the matrix, all the entries from the first nonzero component to the diagonal are stored. These rows are saved in contiguous locations in an one-dimensional array S. An auxiliary vector δ of length N is used to locate the positions of the diagonal components in the primary storage S.

The scheme requires $|Env(A)|+N$ primary locations and N extra pointers. The example in figure 2.2 illustrates Jennings' storage scheme. In view of theorem 2.1, the storage scheme is attractive in the context of symmetric factorization. The factorization can be done in place: each row of the matrix in the primary storage

$$A_{i,f_i(A)}, \ldots, A_{i,i-1}, A_{i,i}$$

can be replaced by the corresponding row of the triangular factor

$$L_{i,f_i(A)}, \ldots, L_{i,i-1}, L_{i,i}.$$

By keeping the auxiliary array δ, we have a mapping function that allows us to access any nonzero component efficiently. The one-to-one mapping from Env(A) to $\{1, 2, \ldots, |Env(A)|\}$ is given by:

$$\{i,j\} \longmapsto \delta_i - (i-j).$$

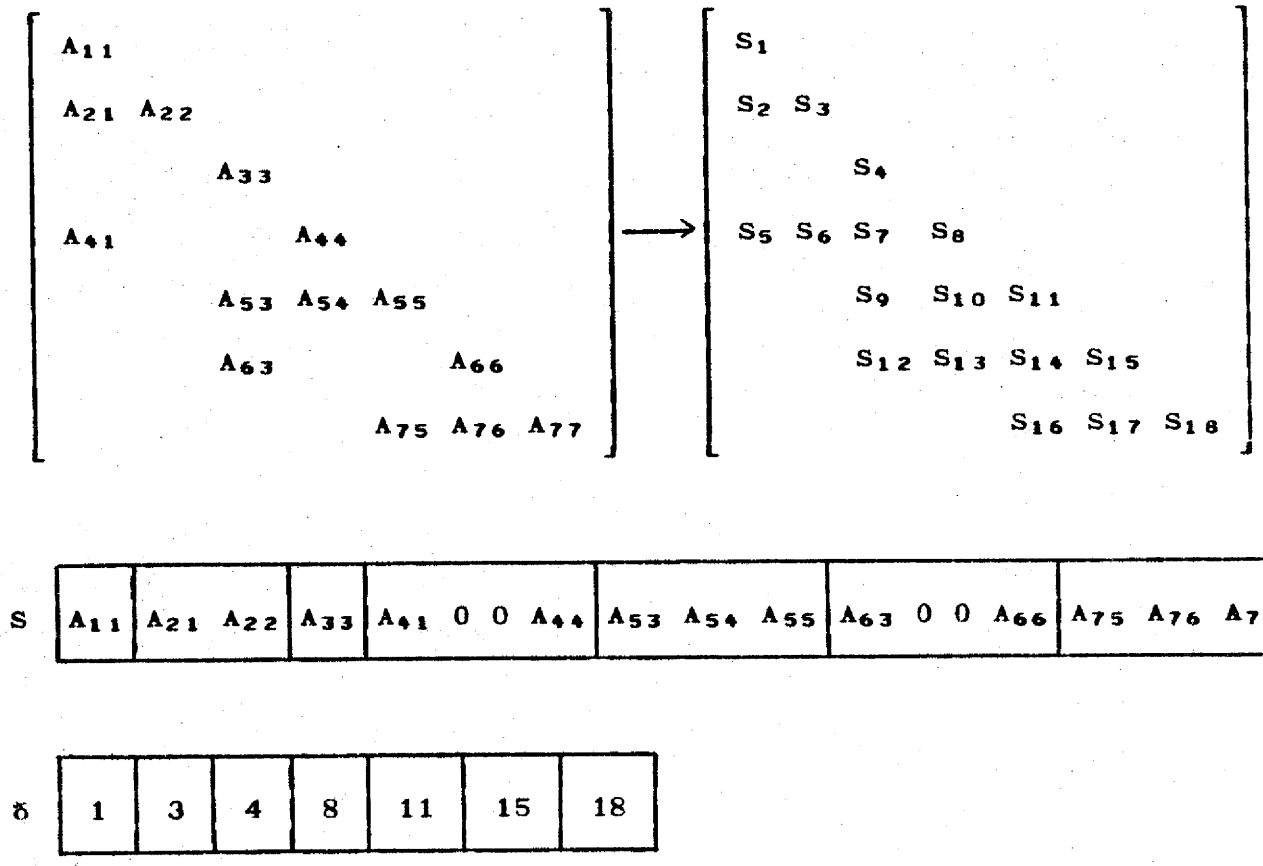


Figure 2.2: Jennings' storage scheme

Thus, a component A_{ij} within the envelope region of A is stored in $S(\delta_i - i + j)$ in the primary storage.

Furthermore, the data organization allows rapid row operations on the envelope of the matrix. To access the i-th row of the lower triangle of the matrix, it is sufficient to have quantities δ_{i-1} and δ_i. Indeed, it is given by:

$$0, \ldots, 0, S(\delta_{i-1}+1), \ldots, S(\delta_i - 1), S(\delta_i).$$

For a compactly stored sparse matrix, it is important to properly relate the order of storage and the order of operations on the stored entries. In the next section, we will see how the row operations in the factorization process can be performed effectively using this storage method.

2.3 Envelope Factorization

Consider the symmetric factorization LL^T of an N by N symmetric positive definite matrix A. Assume the matrix is stored using the Jennings' envelope scheme. Since the storage scheme is row-oriented, it is more efficient to implement the factorization in a row by row manner. In this way, zeros to the left of the first nonzero in each row can be exploited more conveniently in the factorization.

In the row by row determination, the process can be defined by the following equations. For row i (i=1,2,...,N),

$$L_{ij} = \{A_{ij} - \sum_{k=\max\{f_i,f_j\}}^{j-1} L_{ik}L_{jk}\}/L_{jj}, \quad j=f_i,\ldots,i-1,$$

$$L_{ii}^2 = A_{ii} - \sum_{k=f_i}^{i-1} L_{kk}^2. \qquad (2.9)$$

Note that (2.9)† is the inner-product formulation of the factorization process, where zeros outside the envelope region are exploited in computing the inner product $\sum L_{ik}L_{jk}$.

To measure the amount of arithmetic involved, we let $\theta_E(A)$ represent the number of multiplicative operations required to effect the symmetric envelope factorization of the matrix A as given by (2.9). If $\theta(A)$ is the operation

† When the matrix A is clear from the context, we use f_i and β_i instead of $f_i(A)$ and $\beta_i(A)$.

count when all zeros in the matrix are exploited, clearly we have:

$$\theta(A) \le \theta_E(A).$$

We now establish a bound for $\theta_E(A)$ in terms of the varying bandwidths.

<u>Lemma 2.2</u>: $$\theta_E(A) \le \tfrac{1}{2} \sum_{i=1}^{N} \beta_i(A)\,[\beta_i(A)+3]. \qquad (2.10)$$

<u>Proof</u>: Consider equation (2.9). To compute L_{ij}, we need $j-\max\{f_i, f_j\}+1$ operations, which is no larger than $j-f_i+1$. Summing over j, we get

$$\sum_{j=f_i}^{i-1} (j-f_i+1) = \tfrac{1}{2}(i-f_i)\,(i-f_i+1).$$

On the other hand, the diagonal component L_{ii} requires $i-f_i$ multiplications. Hence for the i-th row of L, the number of arithmetic operations required is no larger than

$$\tfrac{1}{2}(i-f_i)\,(i-f_i+1) + i-f_i = \tfrac{1}{2}(i-f_i)\,(i-f_i+3).$$

Recalling that $\beta_i = i-f_i$, we have the desired inequality (2.10). □

It should be noted that for a general profile-oriented matrix, the upperbound in lemma 2.2 is often an over-estimate. For a necessary and sufficient condition that (2.10) is an exact count, we introduce monotone-profile matrices. A symmetric matrix A is said to satisfy the <u>monotone profile</u> property if

$$f_j(A) \le f_i(A), \quad \text{for } j \le i. \qquad (2.11)$$

Theorem 2.2: $$\theta_E(A) = \frac{1}{2}\sum_{i=1}^{N} \beta_i(A)\,\{\beta_i(A)+3\}$$

if and only if the matrix A has the monotone profile property.

Proof: From the proof of lemma 2.2, we note that (2.10) is an equality if and only if $\max\{f_i, f_j\} = f_i$ for $j \le i$, which is equivalent to the monotone profile property of (2.11). □

For an exact operation count for a general symmetric matrix, it is useful to introduce the frontwidth concept. For the i-th row of the matrix A, we let

$$w_i(A) = |\{k: k>i \text{ and } A_{kl}\neq 0 \text{ for some } l\le i\}|. \quad (2.12)$$

The number $w_i(A)$ is simply the number of "active" rows at the i-th step of factorization; that is, the number of rows in the envelope of A, which intersect column i. The quantity

$$w(A) = \max\{w_i(A): 1\le i\le N\}, \quad (2.13)$$

is usually referred to as the wavefront or frontwidth of the matrix A ([George73b], [Irons70], [Melosh69]).

Consider the matrix in figure 2.1. Its i-th frontwidth $w_i(A)$ is given in figure 2.4.

The fact that the frontwidth concept is relevant to the analysis of the envelope method is justified by the following observations.

$$A = \begin{bmatrix} x & x & & x & & & \\ x & x & & \blacksquare & & & \\ & & x & o & x & x & \\ x & \blacksquare & o & x & x & o & \\ & & x & x & x & \blacksquare & x \\ & & x & o & \blacksquare & x & x \\ & & & & x & x & x \end{bmatrix}$$

i	$w_i(A)$	$\beta_i(A)$
1	2	0
2	1	1
3	3	0
4	2	3
5	2	2
6	1	3
7	0	2

Figure 2.4: Wavefront of an 7 by 7 matrix.

<u>Lemma 2.3</u>: $|Env(A)| = \sum_{i=1}^{N} w_i(A)$. □

<u>Lemma 2.4</u>: $\theta_E(A) = \frac{1}{2} \sum_{i=1}^{N} w_i(A) \{w_i(A)+3\}$.

<u>Proof</u>: If we treat the envelope of A as full, the number of nonzeros in L_{*i} is $w_i(A)+1$. The result then follows from lemma 1.1. □

2.4 Envelope Substitution

Given the lower triangular factor L in envelope form, it is straightforward to perform the forward and backward substitution:

$$L\,y = b, \tag{2.14}$$

$$L^T x = y. \tag{2.15}$$

The forward solution can be defined by the equations: for $i=1,\ldots,N$,

$$y_i = \{b_i - \sum_{k=f_i}^{i-1} L_{ik} y_k\}/L_{ii}. \tag{2.16}$$

The row by row envelope storage scheme is well suited for this computation. The task of performing the backward substitution (2.15) is equally simple. The defining equations are: for $i=N,N-1,\ldots,1$,

$$x_i = \{y_i - \sum_{k=i+1}^{N} L_{ki} x_k\}/L_{ii}, \tag{2.17}$$

where the summation is taken only over those k with $f_k(A) \leq i$. Due to the inherent nature of the data sequence, it is more convenient to rearrange the order in which the inner products $\Sigma\, L_{ki} x_k$ are computed. It can be described as follows.

```
procedure BACKWARD_SOLVE(N, x, y, L);
   for i:= 1 step 1 until N do
      x_i:= y_i;

   for i:= N step -1 until 1 do
   begin
      x_i:= x_i/L_ii;
      for j:= f_i step 1 until i-1 do
         x_j:= x_j - L_ij x_i;
   end;
end;
```

Lemma 2.5: The forward and backward envelope substitution can be done in $2(|Env(A)|+N)$ multiplicative operations.□

In the entire solution process, note that the diagonal components L_{ii} are only used as divisors. Since on most computers division is slower than multiplication, it is preferrable to store the reciprocal of the L_{ii}; that is, in the storage scheme, $S(\delta_i)$ will contain L_{ii}^{-1}.

2.5 Effect of Reordering

Consider the direct solution of an N by N sparse symmetric positive definite system A x = b. The exact storage and computational requirements for its envelope solution are given in lemmas 2.3 and 2.4 respectively. These quantities depend upon the locations (2.1) of leading nonzeros in the rows of A. These locations can, however, be altered by appropriately rearranging the equations and renumbering the unknowns. It is sometimes possible to find a permutation matrix P such that the envelope of the permuted matrix PAP^T is much smaller than that of A.

$$A = \begin{bmatrix} x & & & & & \\ x & x & & \text{sym} & & \\ x & \blacksquare & x & & & \\ x & \blacksquare & \blacksquare & x & & \\ x & \blacksquare & \blacksquare & \blacksquare & x & \\ x & \blacksquare & \blacksquare & \blacksquare & \blacksquare & x \end{bmatrix} \qquad PAP^T = \begin{bmatrix} x & & & & & \\ & x & & \text{sym} & & \\ & & x & & & \\ & & & x & & \\ & & & & x & \\ x & x & x & x & x & x \end{bmatrix}$$

Figure 2.5: Ordering for a small envelope.

Consider the matrix example in figure 2.5. It is the matrix of the well-known *star graph*. Let P be the permutation matrix that reverses the ordering in A. It can be readily verified that $|Env(A)| = 15$, $\theta_E(A) = 50$; and $|Env(PAP^T)| = 5$, $\theta_E(PAP^T) = 10$. Generalizing the sparse

structure to a matrix of N nodes, we have

$$\frac{|\mathrm{Env}(A)|}{|\mathrm{Env}(PAP^T)|} = \frac{\frac{1}{2}N(N-1)}{N-1} \simeq \frac{N}{2},$$

$$\frac{\theta_E(A)}{\theta_E(PAP^T)} = \frac{\frac{1}{2}\sum i(3+i)}{\frac{1}{2}\sum 4} \simeq \frac{N^2}{3}.$$

This example shows some dramatic savings. For practical applications, the amount saved is often quite substantial so that it is usually worthwhile to preorder the system before the symmetric factorization of the matrix is performed. In the next chapter, the reordering problem will be studied from a combinatorial point of view.

Chapter 3 Graph Theoretic Study of the Envelope Method

Graph theory is an elegant branch of modern mathematics that has a wide variety of applications. It provides simple and yet powerful mathematical tools "for solving problems having to do with discrete arrangements of objects." ([Busacker65]) Indeed, it is a convenient vehicle in the study of reordering problems for sparse matrices. In this chapter, we will examine the reordering problem with respect to the envelope method using graph models. Graph theoretic interpretations of the relevant matrix definitions in chapter 2 are given. In these connection, we define minimal and minimum envelope orderings. Some preliminary results on these orderings are established for future use.

3.1 Notations from Graph Theory

We begin this section with some standard graph theoretic terminology.

A graph $G=(X,E)$ consists of a finite set of nodes together with a set E of unordered pairs of nodes called edges. A graph $\bar{G}=(\bar{X},\bar{E})$ is a subgraph of $G=(X,E)$ if $\bar{X} \subset X$ and $\bar{E} \subset E$; G is then a super graph of $\bar{G}$. For a subset $Y \subset X$, the section graph determined by Y is the subgraph (Y,E'), where

$$E' = \{ \{x,y\} \in E: x, y \in Y \}.$$

Sometimes, we use "the subgraph Y" to refer to the section graph determined by the node subset Y.

Two nodes x and y are said to be <u>adjacent</u> if $\{x,y\} \in E$. For a subset $Y \subset X$, the <u>adjacent set</u> of Y is defined as

$$Adj(Y) = \{x \in X \backslash Y: \{x,y\} \in E \text{ for some } y \in Y\}. \quad (3.1)$$

In case $Y = \{y\}$, we shall use Adj(y) instead of the formally correct Adj({y}). The <u>degree</u> of a node x is the number of nodes adjacent to x, denoted by deg(x). Sometimes, we refer to a node $y \in Adj(x)$ as a <u>neighbor</u> of x. A <u>clique</u> is a maximal subgraph whose nodes are pairwise adjacent. When the graph is itself a clique, it is said to be <u>complete</u>.

An edge $e \in E$ is said to be <u>incident</u> at a node x if $x \in e$. For a subset $Y \subset X$, the <u>incidence set</u> of Y ([Mayeda72]) is

$$Inc(Y) = \{ \{x,y\} \in E: y \in Y, x \in X \backslash Y \}. \quad (3.2)$$

Again, when $Y = \{y\}$, we shall write Inc(y).

In the graph of figure 3.1, $Adj(x_5) = \{x_3, x_4, x_7\}$, $Inc(x_5) = \{ \{x_4,x_5\}, \{x_3,x_5\}, \{x_5,x_7\} \}$. If $Y=\{x_2, x_3, x_5\}$, then $Adj(Y) = \{x_1, x_4, x_6, x_7\}$ and $Inc(Y) = \{ \{x_1,x_2\}, \{x_3,x_6\}, \{x_4,x_5\}, \{x_5,x_7\} \}$.

A <u>path</u> of <u>length</u> l is a sequence of $l \geq 1$ edges $\{x_0,x_1\}, \{x_1,x_2\}, \ldots, \{x_{l-1},x_l\}$, where all nodes on the path, except possibly x_0 and x_l, are distinct. A <u>cycle</u> is a path, which begins and ends at the same node. A graph G is

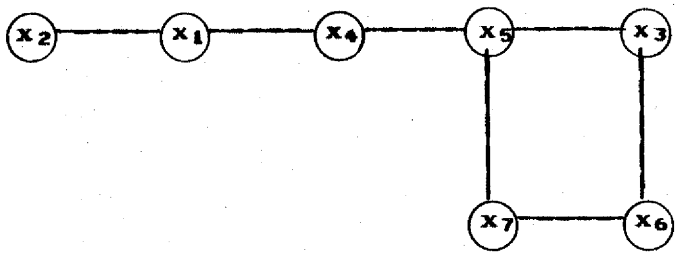


Figure 3.1 A graph with 7 nodes

connected if for each pair of distinct nodes, there is a path connecting them. If G is not connected, it consists of two or more maximal connected subgraphs called components.

Let G=(X,E) be a connected graph. The distance d(x,y) between two nodes x and y in G is the length of a shortest path connecting them. Following Berge [Berge58], we define the eccentricity of a node x to be the quantity

$$l(x) = \max \{d(x,y): y \in X\}. \qquad (3.3)$$

The diameter of G is then defined as

$$\delta(G) = \max \{l(x): x \in X\}; \qquad (3.4)$$

or equivalently,

$$\delta(G) = \max \{d(x,y): x,y \in X\}.$$

A node $x \in X$ is said to be a peripheral node if its eccentricity l(x) is the diameter of the graph. Consider the example in figure 3.1. The diameter of the graph is 5; x_2 and x_6 are the only peripheral nodes.

The concept of level structure was first introduced by Arany et al [Arany71] in their study of bandwidth reduction algorithms. Formally, a <u>level structure</u> of a graph G=(X,E) is a partition

$$\mathcal{L} = \{L_0, L_1, \ldots, L_l\}$$

of the node set X such that for i=1,...,l-1,

$$Adj(L_i) \subset L_{i-1} \cup L_{i+1}. \qquad (3.5)$$

It then follows from (3.5) that for a level structure,

$$Adj(L_0) \subset L_1,$$

$$Adj(L_l) \subset L_{l-1}.$$

The number l is the <u>length</u> of the level structure $\mathcal{L}$, while the quantity

$$\max \{|L_i|: 0 \le i \le l\} \qquad (3.6)$$

is called the <u>width</u> of $\mathcal{L}$.

A particularly important class of level structures is those that are rooted. For a node $x \in X$, the <u>rooted level structure</u> at x is defined as the level structure:

$$\mathcal{L}(x) = \{L_0(x), L_1(x), \ldots, L_{l(x)}(x)\}, \qquad (3.7)$$

where $L_0(x) = \{x\}$,

$$L_i(x) = Adj\Big(\bigcup_{k=0}^{i-1} L_k(x)\Big), \qquad i=1,\ldots,l(x).$$

In [Tutte67], (3.7) is also referred to as a <u>recessional sequence</u>. It should be noted that the length of the rooted level structure $\mathcal{L}(x)$ is exactly the eccentricity l(x) of the node x. The rooted level structure at x_1 of the graph in figure 3.1 is given in figure 3.2. The length of $\mathcal{L}(x_1)$ is 4 and its width is 2.

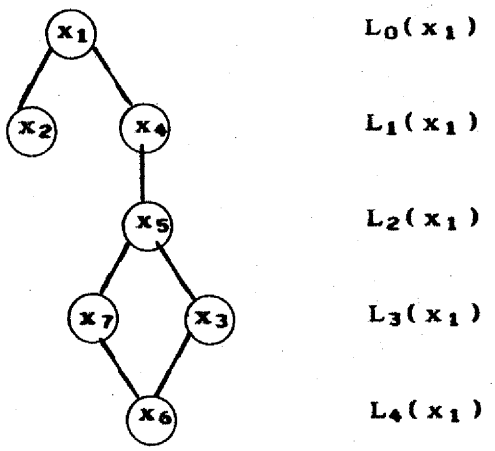


Figure 3.2 Rooted level structure at x_1

For a graph G=(X,E) with N nodes, a <u>labelling</u> or <u>ordering</u> of G is a bijective mapping α:{1,2,...,N}--->X. We will use G_α and X_α to denote the labelled graph and the labelled node set respectively.

3.2 Graph Theoretic Study of Envelope Structure

Let A be an N by N symmetric matrix. To study the envelope method graph-theoretically, we associate an undirected graph G(A) = (X(A),E(A)) with the matrix A. Here X(A) is the set of nodes

$$\{a_1, a_2, \ldots, a_N\},$$

where a_i corresponds to the i-th row of A, and E(A) is the set of edges where $\{a_i,a_j\} \in E(A)$ if and only if the component A_{ij} is nonzero and $i \neq j$. Note that G(A) has an implicit labelling defined by the matrix, which maps the integer i to the node a_i. There is clearly a one-to-one correspondence between labelled graphs with N nodes and N by N symmetric binary matrices with unit diagonal.

The undirected graph in figure 3.1 is associated with the symmetric matrix in figure 2.1. We present them together in figure 3.3 for reference.

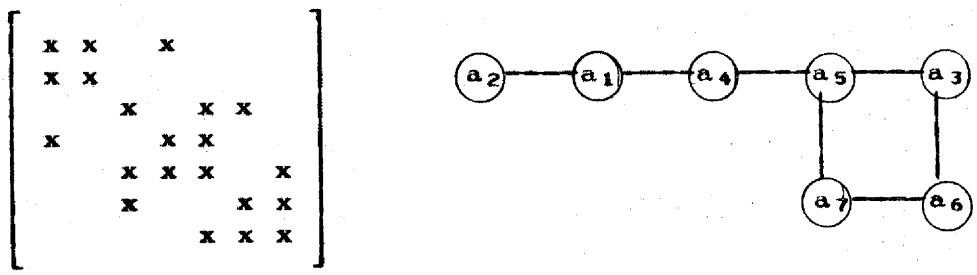


Figure 3.3: Matrix and its associated graph

It is appropriate here to point out that each labelling α on G(A) identifies uniquely a permutation matrix P_α on A such that

$$G(A)_\alpha = G(P_\alpha A P_\alpha^T).$$

The labelled graphs G(A) and $G(P_\alpha A P_\alpha^T)$ are isomorphic, but the node labels in the latter have been permuted according to P_α. So, the reordering problem can be studied conveniently using the underlying structure of G(A).

Since we are going to study our problem from a combinatorial point of view, it is important to give graph theoretic interpretations of the various matrix definitions introduced in sections 2.1 and 2.3. In this section, we redefine these concepts for labelled graphs. Although they are defined in terms of the graph structure, the relationships to their matrix counterparts should be obvious.

We first consider the set Fill(A) (2.8) in the symmetric factorization of the matrix A. This set can be described conveniently in terms of the associated graphs:

$$G(A) = (X(A), E(A)),$$

and $$G(L+L^T) = (X(L+L^T), E(L+L^T)),$$

where $A = LL^T$. It should be noted that

$$X(A) = X(L+L^T) = \{a_1, \ldots, a_N\}.$$

and that G(A) is a subgraph of $G(L+L^T)$.

<u>Lemma 3.1</u> ([Parter61]): The unordered pair $\{a_i, a_j\} \in E(L+L^T)$ if and only if $\{a_i, a_j\} \in E(A)$ or $\{a_i, a_k\}$, $\{a_j, a_k\} \in E(L+L^T)$ for some $k < \min\{i,j\}$. □

Theorem 3.1: Let $j < i$. The unordered pair $\{a_i, a_j\} \in E(L+L^T)$ if and only if there is a path connecting a_i and a_j in the section graph of $G(A)$ determined by $\{a_1, \ldots, a_j\} \cup \{a_i\}$.

Proof The "if" part follows from lemma 3.1 and an induction on the length of the connecting path between a_i and a_j. On the other hand, an induction on the subscript j with lemma 3.1 proves the "only if" part. □

Let $G_\alpha = (X_\alpha, E)$ be a graph with labelling $\alpha: \{1,2,\ldots,N\} \longrightarrow X$. For convenience, we denote $\alpha(i)$ by x_i. In order to introduce the fill of G_α without any matrix notion, we use the result of theorem 3.1 and define the super-graph:

$$G[\alpha] = (X, E[\alpha]), \tag{3.9}$$

where $\{x_i, x_j\} \in E$ $(j<i)$ if and only if there is a path from x_i to x_j in $\{x_1, \ldots, x_j\} \cup \{x_i\}$. The fill of G_α can then be defined to be

$$\text{Fill}(G_\alpha) = E[\alpha] \setminus E. \tag{3.10}$$

In [Rose70], $\text{Fill}(G_\alpha)$ is called a **triangulation set** for G, since these edges when added to E transform G into a triangulated† graph $G[\alpha]$. Rose refers to the graph $G[\alpha]$ as the **monotone transitive extension** of G_α. This extension graph has the interesting property that

$$\text{Fill}(G[\alpha]_\alpha) = \emptyset.$$

---

† A graph is **triangulated** if every cycle in it has a chord. [Berge58]

The following properties of the function $w_i(G_\alpha)$ can be readily verified using lemmma 3.3.

Lemma 3.4: $w_i(G_\alpha) \leq w_{i+1}(G_\alpha) + 1, \quad 1 \leq i \leq N-1$

and $w_N(G_\alpha) = 0.$

Proof: It follows from the relation

$$Adj(\{x_1,\ldots,x_i\}) \subset Adj(\{x_1,\ldots,x_i,x_{i+1}\}) \cup \{x_{i+1}\},$$

where the union on the right hand side is disjoint. □

Lemma 3.5: The graph G is connected, if and only if $w_i(G_\alpha) \neq 0$, for $1 \leq i \leq N-1$. □

Consider the graph in figure 3.3. We have

$$Adj(\{a_1\}) = \{a_1, a_4\},$$

$$Adj(\{a_1, a_2\}) = \{a_4\},$$

$$Adj(\{a_1, a_2, a_3\}) = \{a_4, a_5, a_6\},$$

$$Adj(\{a_1, \ldots, a_4\}) = \{a_5, a_6\},$$

$$Adj(\{a_1, \ldots, a_5\}) = \{a_6, a_7\},$$

$$Adj(\{a_1, \ldots, a_6\}) = \{a_7\},$$

and $$Adj(\{a_1, \ldots, a_7\}) = \emptyset.$$

To complete the set of definitions, we let $\theta_E(G_\alpha)$ to denote the quantity

$$\frac{1}{2} \sum_{i=1}^{N} w_i(G_\alpha) \{w_i(G_\alpha)+3\},$$

which corresponds to the number of factorization operations when $G_\alpha = G(A)$ for some symmetric matrix A.

3.3 Minimial and Minimum Envelope Orderings

In his analysis of the elimination process [Rose72a], Rose uses the notion of a triangulation set (3.10) to formulate the minimal and minimum triangulation (or fill) problems. For a graph $G = (X,E)$, the former requires the finding of a labelling α such that for any other β,

$$Fill(G_\beta) \subset Fill(G_\alpha) \Longrightarrow Fill(G_\beta) = Fill(G_\alpha). \quad (3.14)$$

The second problem involves the finding of an α with a minimum set of fills, that is, for any β,

$$|Fill(G_\alpha)| \leq |Fill(G_\beta)|. \quad (3.15)$$

For efficient algorithms to determine minimal fill orderings, see [Ohtsuki75], [Rose75].

To establish the equivalent problems in envelope methods, we introduce the set of potential fills. Let $G_\alpha = (X_\alpha,E)$ be a graph labelled by α. We define the <u>potential fill</u> of G_α as:

$$Pfill(G_\alpha) = Env(G_\alpha) \setminus E. \quad (3.16)$$

In terms of the matrix A, Pfill(G(A)) corresponds to the set of locations of zero entries in the envelope of the matrix A.

Let α be an ordering on G. Then α is a <u>full envelope ordering</u> if $Pfill(G_\alpha) = \emptyset$. We call α a <u>minimal envelope ordering</u> if for any ordering β,

$$Pfill(G_\beta) \subset Pfill(G_\alpha) \Longrightarrow Pfill(G_\beta) = Pfill(G_\alpha). \quad (3.17)$$

The labelling α is a **minimum envelope ordering** if

$$|Pfill(G_\alpha)| \leq |Pfill(G_\phi)|, \qquad (3.18)$$

for every ordering ϕ.

Clearly, any full envelope ordering gives a minimum envelope; and any minimum envelope ordering is minimal.

The ordering in figure 3.4 generates a full envelope. In figure 3.5, the ordering is a minimum envelope ordering with {{5,3}} as the set of potential fill. Figure 3.6 is a minimal envelope ordering. Here the set of potential fill is {{6,3}, {6,4}}; however, we can find a different ordering with |Pfill| = 1.

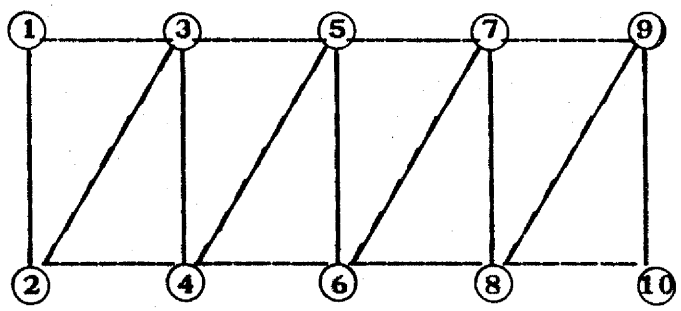


Figure 3.4: A full envelope ordering

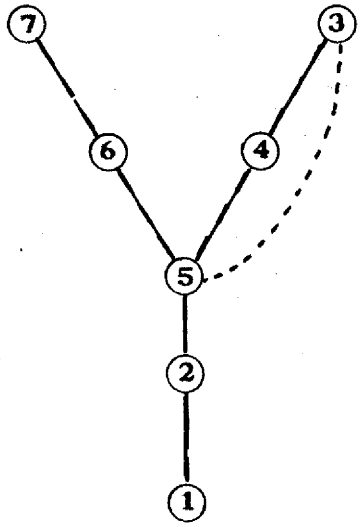


Figure 3.5: A minimum envelope ordering

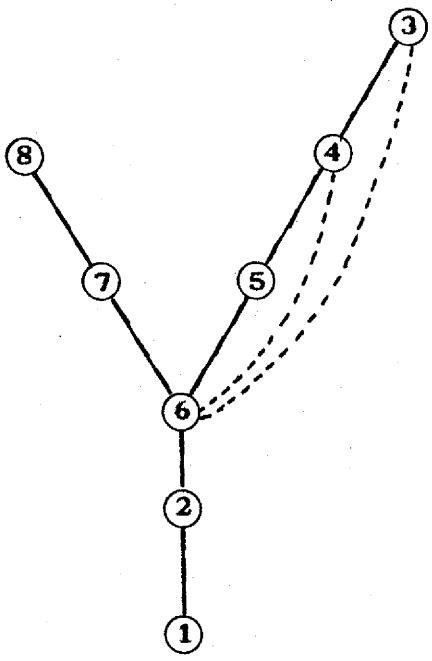


Figure 3.6: A minimal envelope ordering

For $j=i+1,\ldots,k$, the relation on the adjacent sets:

$$\begin{aligned} &Adj(\{y_1,\ldots,y_{i-1},y_i,\ldots,y_j\}) \\ &= Adj(\{x_1,\ldots,x_{i-1},x_k,x_i,\ldots,x_{j-1}\}) \\ &= Adj(\{x_1,\ldots,x_{i-1},x_i,\ldots,x_{j-1}\}) \setminus \{x_k\} \end{aligned}$$

implies that

$$\begin{aligned} w_j(G_{\bar{\alpha}}) &= |Adj(\{x_1,\ldots,x_i,\ldots,x_{j-1}\}) \setminus \{x_k\}| \\ &= w_{j-1}(G_\alpha) - 1. \end{aligned}$$

Using lemma 3.4, we have

$$w_j(G_{\bar{\alpha}}) \le w_j(G_\alpha).$$

It then follows from our claim that

$$|Env(G_{\bar{\alpha}})| \le |Env(G_\alpha)|,$$

and $$\theta_E(G_{\bar{\alpha}}) \le \theta_E(G_\alpha).$$ □

Theorem 3.2 provides some guidelines in the general relabelling problem for profile minimization. Suppose the first i-1 nodes in some ordering have already been numbered, say $\{x_1,\ldots,x_{i-1}\}$. If, for some reasons, we want to choose node x as the i-th one in our ordering, it is never worse, in the context of minimum profile, to find all those unnumbered y in

$$Adj(\{x_1,\ldots,x_{i-1},x\}),$$

where $Adj(\{x_1,..,x_{i-1},y\}) \cup \{y\}$ is properly contained in $Adj(\{x_1,..,x_{i-1},x\}) \cup \{x\}$ and number them before x.

The following corollaries are immediate from theorem 3.2.

Corollary 3.6: Let α and $\bar{\alpha}$ be as in theorem 3.2. If

$$Adj(\{x_1,\ldots,x_{i-1},x_k\}) \cup \{x_k\} \subseteq Adj(\{x_1,\ldots,x_{i-1}\}),$$

then $|Env(G_{\bar{\alpha}})| \leq |Env(G_{\alpha})|$,

and $\theta_E(G_{\bar{\alpha}}) \leq \theta_E(G_{\alpha})$. □

Corollary 3.7: If there is a node y such that

$$Adj(\{x_1,\ldots,x_{i-1},y\}) \cup \{y\} \subsetneq Adj(\{x_1,\ldots,x_{i-1},x_i\}) \cup \{x_i\},$$

then no minimum envelope ordering for G can begin with the numbering $x_1,\ldots,x_{i-1},x_i$. □

3.4 On Full Envelope Orderings

Except in some rather special cases, a graph does not possess full envelope orderings. Yet, it is still worthwhile to study them for two reasons. Firstly, if α is an ordering on a graph $G=(X,E)$, then α will be a full envelope ordering on the super graph

$$(X, \mathrm{Env}(G_\alpha)).$$

Secondly, they do bear some significance in the study of minimal envelope orderings. If $\bar{\alpha}$ is a minimal envelope ordering on G, then for any nonempty subset $H \subseteq \mathrm{Pfill}(G_{\bar{\alpha}})$, the super graph

$$(X, \mathrm{Env}(G_{\bar{\alpha}})\backslash H)$$

does not have any full envelope ordering.

To begin, we give a straightforward characterization of full envelope orderings.

Lemma 3.8: Let α be a labelling for the graph G, and let $x_i = \alpha(i)$. The labelling α is a full envelope ordering if and only if

$$\mathrm{Adj}(\{x_1,\ldots,x_i\}) \subseteq \mathrm{Adj}(x_i), \qquad i=1,\ldots,N.$$

Proof: From lemma 3.2, $\{x_i,x_j\} \in \mathrm{Pfill}(G_\alpha)$ if and only if $x_j \in \mathrm{Adj}(\{x_1,\ldots,x_i\})$ and $\{x_i,x_j\} \notin E$. Thus, $\mathrm{Pfill}(G_\alpha)=\emptyset$ if and only if for $1\le i\le N$, $\mathrm{Adj}(\{x_1,\ldots,x_i\}) \subseteq \mathrm{Adj}(x_i)$. □

Another characterization in terms of a sequence of complete graphs is given in the following lemma.

Lemma 3.9: Let α be a labelling for G, and $x_i = \alpha(i)$. The labelling α is a full envelope ordering if and only if $Adj(\{x_1,\ldots,x_i\}) \cup \{x_i\}$ is complete for every $1 \le i \le N$.

Proof: "if part" By lemma 3.8, it is sufficient to show that $Adj(\{x_1,\ldots,x_i\}) \subset Adj(x_i)$. Consider any $y \in Adj(\{x_1,\ldots,x_{i-1}\}) \setminus \{x_i\}$. Since $Adj(\{x_1,\ldots,x_{i-1}\}) \cup \{x_i\}$ is complete, y also belongs to $Adj(x_i)$.

"only if part" We first show that for i=1,...,N, $Adj(\{x_1,\ldots,x_i\})$ is complete. Assume for contradiction, for some i, $Adj(\{x_1,\ldots,x_i\})$ is not. Then, there exist

$$x_j,\ x_k \in Adj(\{x_1,\ldots,x_i\})$$

such that x_j and x_k are not adjacent. For definiteness, let $j > k$. Then

$$x_j \in Adj(\{x_1,\ldots,x_i,\ldots,x_k\})$$

and yet $x_j \notin Adj(x_k)$.

To complete the proof, we show $Adj(\{x_1,\ldots,x_i\}) \cup \{x_i\}$ is a complete graph. If it is not, since $Adj(\{x_1,\ldots,x_i\})$ is, we can always find an x in $Adj(\{x_1,\ldots,x_i\})$ such that x does not belong to $Adj(x_i)$. □

Lemmas 3.8 and 3.9 are characterizations of full envelope orderings on the ordered graph G_α. In the remaining part of the section, we will study the unordered structure of graphs with full envelope orderings.

Let $G=(X,E)$ be a connected graph. A separator V of G is a subset of nodes whose removal disconnects the graph. A separator is minimal if no subset of it is a separator. If $\{x\}$ is a separator, then the node x is said to be a cutnode. Clearly, cutnodes are minimal separators. In the graph of figure 3.1, $\{x_3,x_7\}$ is a minimal separator, while x_5 is a cutnode.

Consider a minimal separator V of a connected graph $G=(X,E)$. Let the removal of V define k components $C_i=(X_i,E_i)$, $1\leq i\leq k$. The following two lemmas are immediate.

Lemma 3.10: For $1\leq i\leq k$, if $Y \subset X_i$, then $Adj(Y) \subset V \cup X_i$. Furthermore, if $Y \neq X_i$, $Adj(Y) \cap X_i \neq \emptyset$. □

Lemma 3.11: For $1\leq i\leq k$, $X_i \cap Adj(v) \neq \emptyset$ for every $v \in V$.

Proof For contradiction, let $v \in V$ where $X_i \cap Adj(v)=\emptyset$. Then the subset $V' = V\setminus\{v\}$ of V still separates X_i from $X\setminus(X_i \cup V')$. This contradicts the minimality of V. □

A full envelope ordering resequences the nodes in G according to some pattern with respect to the components of minimal separators. It is given in the following lemma.

Lemma 3.12: Let V be a minimal separator of G with k components $C_i=(X_i,E_i)$. If α is a full envelope ordering on G, then there exists a permutation ϕ on the components so

that α orders the nodes in the sequence:

$$X_{\phi(1)}, X_{\phi(2)}, \ldots, X_{\phi(k-1)}, V \cup X_{\phi(k)}.$$

Proof: Let k=2. Because of lemma 3.11, the nodes in V cannot be numbered until there is only one or part of one component left unnumbered, say $C_{\phi(2)}$.

Let $x_1 \in X_{\phi(1)}$. If $n_1 = |X_{\phi(1)}|$, we show that $x_1, \ldots, x_{n_1} \in X_{\phi(1)}$. Assume that $x_1, \ldots, x_{i-1} \in X_{\phi(1)}$ where $i \leq n_1$. By lemma 3.9, x_i has to be chosen so that $Adj(\{x_1, \ldots, x_{i-1}\}) \cup \{x_i\}$ is complete. By lemma 3.10,

$$Adj(\{x_1, \ldots, x_{i-1}\}) \cap X_{\phi(1)} \neq \emptyset$$

so x_i cannot be in $X_{\phi(2)}$. Thus $x_i \in X_{\phi(1)}$. This means α orders the two components in some sequence ϕ

$$X_{\phi(1)}, V \cup X_{\phi(2)}.$$

The proof can then be completed by induction on k, the number of components. □

Theorem 3.3: Let G be a graph that has full envelope orderings. For every minimal separator V of G with components $C_i=(X_i,E_i)$, $1 \leq i \leq k$, the set $V \cup \{x\}$ is complete for every $x \in X_i$, for all except possibly two components.

Proof: Let α be a full envelope ordering on G. Lemma 3.12 assures that α orders the nodes in some component sequence: $X_{\phi(1)}, X_{\phi(2)}, \ldots, X_{\phi(k-1)}, V \cup X_{\phi(k)}$.
Consider any $x \in X_{\phi(2)} \cup \ldots \cup X_{\phi(k-1)}$. Let $\alpha(j)=x \in X_{\phi(i)}$.

By lemma 3.11, $Adj(X_{\phi(1)} \cup \ldots \cup X_{\phi(i-1)}) = V$, so that $V \subset Adj(\{\alpha(1), \ldots, \alpha(j-1)\})$. The result of lemma 3.9 implies that $V \cup \{x\}$ is complete. □

We shall find theorem 3.3 useful in chapter 4 when a profile algorithm for trees is analysed. In the proof of theorem 3.3, it is important to realize that the components $C_{\phi(i)}$, $2 \leq i \leq k-1$, when considered as independent graphs, have full envelope orderings. Thus, the result in theorem 3.3 is also applicable to these components.

3.5 On Maximal Envelope Fill Orderings

Let $G=(X,E)$ be a graph labelled by α. The labelled graph G_α is said to suffer **maximal envelope fill** if $Fill(G_\alpha) = Pfill(G_\alpha)$. In this section, we give simple sufficient conditions on G_α so that it suffers maximal fill.

Theorem 3.4: Let $G_\alpha=(X_\alpha,E)$, with $X=\{x_1,\ldots,x_N\}$. If the subgraphs $\{x_1,\ldots,x_i\}$, $1\le i\le N$ are connected, then G_α suffers maximal envelope fill.

Proof: Consider $\{x_i,x_j\} \in Pfill(G_\alpha)$, with $j < i$. By definitions (3.16), (3.11), $\{x_i,x_{f_i}\} \in E$ and $f_i(G_\alpha) \le j$. Since $\{x_1,\ldots,x_j\}$ is connected, we can find a path from x_{f_i} to x_j. Together with the edge $\{x_i,x_{f_i}\}$, a path from x_i to x_j exists in the section graph $\{x_1,\ldots,x_j\}\cup\{x_i\}$. By definition (3.10), $\{x_i,x_j\} \in Fill(G_\alpha)$. □

The sufficient condition in theorem 3.4 has an interesting equivalent form.

Lemma 3.13: Let $X=\{x_1,\ldots,x_N\}$. The section graphs $\{x_1,\ldots,x_i\}$, $1\le i\le N$, are connected if and only if $f_i(G_\alpha) < i$ for $1<i\le N$.

Proof: The "if" part follows by an induction on i, while the "only if" part follows from definition of $f_i(G_\alpha)$. □

Theorem 3.5: If G_α has the **monotone profile property** (cf (2.11)):

$$f_j(G_\alpha) \leq f_i(G_\alpha), \quad \text{for } j \leq i, \tag{3.19}$$

then the envelope fill in G_α is maximal.

Proof: Consider the case when G is connected. We show that $f_i(G_\alpha) < i$ for $1 < i \leq N$. Assume for some $i > 1$, $f_i(G_\alpha) = i$. The monotone profile property implies that $f_k(G_\alpha) \geq i$ for $k \geq i$. That is, $\{\alpha(1),..,\alpha(i-1)\}$ and $\{\alpha(i),..,\alpha(N)\}$ are not connected, which contradicts our assumption. By lemma 3.13 and theorem 3.4, G_α suffers maximal envelope fill.

When G is not connected, it follows from the monotone profile property that α orders the connected components of G one after another. The result in the connected case applies to the components of G so that the envelope fill is also maximal. □

Maximal envelope fill orderings have some important implications. In the direct solution of the linear system (1.1), if the associated graph G(A) has maximal envelope fill, the Jennings' envelope storage scheme (see section 2.2) is highly appropriate since we know a priori that only nonzeros will be stored. A matrix formulation of the above results can be found in [George75b] by George and Liu. In chapter 5, we will show that some popular profile orderings exhibit this maximal fill property.

hapter 4: On Reducing the Profile of Trees

In this chapter, we consider the envelope problems for tree structures. The familiar postorder traversal of rooted trees turns out to be an effective algorithm for reducing the profiles of trees. The postordering is modified to an $O(N\log_2 N)$ algorithm MET which always generates a minimal envelope for trees of N nodes. The resulting profile is shown to be bounded by $N+N\log_2 N$.

1.1 Trees

A **tree** is a connected graph with no cycles. The following equivalent definitions for a tree are well known [Berge58].

Theorem 4.1: Let $T=(X,E)$ be a graph. The following statements are equivalent.

i- T is a tree;

ii- T is connected and $|X|=|E|+1$;

iii- every pair of distinct nodes in T is connected by exactly one path. □

A **rooted tree** is an ordered pair (R,T), where R is a distinguished node in the tree $T=(X,E)$. The node R is called the **root**.

Consider the path from the root R to a node x. According to (iii) of theorem 4.1, the path is necessarily

unique. If this path passes through a node y, then y is an ancestor of x and x a descendant of y. If, in addition, $\{x,y\} \in E$, y is called the father of x and x a son of y. Note that a node has only one father node and it can have any number of sons. A node together with all its descendants are called a subtree of the rooted tree. We remark that the ancestor-descendant and the father-son relationships and the subtree concepts are not defined in unrooted trees.

4.2 Postorder of Rooted Trees

Ordering of tree structures in the context of solving symmetric linear systems was first studied by Parter [Parter61]. He shows

Theorem 4.2: Let T=(X,E) be a tree. Then there exists a labelling α on T such that Fill(T_α) is empty. □

Such labellings can be obtained by the following observation.

Theorem 4.3: Fill(T_α) = $\emptyset$ if $\alpha^{-1}(x) < \alpha^{-1}(y)$ whenever y is the father of the node x in the rooted tree (α(N),T). □

However, not all trees have full envelope orderings. As a matter of fact, only those trees with a main "stem" and "branches" of length one can be ordered with full envelopes. This can be readily proved by theorem 3.3. An example of such trees is given in figure 4.2.

A thorough study of tree orderings has been given in [Knuth68] in the context of tree traversals. The book by Aho et al [Aho74] also contains a comprehensive treatment of the subject. In a complete traversal of a given tree, each node is visited exactly once so that it induces a linear arrangement of the nodes. It is apparent that the postorder traversal of rooted trees is most appropriate for profile reduction.

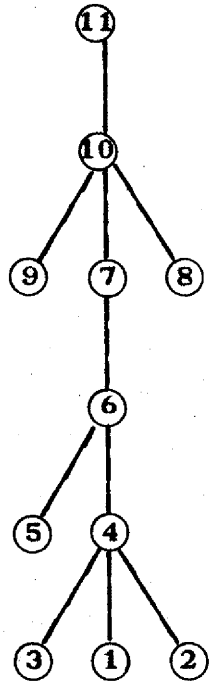


Figure 4.2: A tree with full envelope orderings

Let us first review the postorder traversal of a rooted tree (R,T). Let the rooted subtrees under R be $(s_i(R),T_i(R))$, for $1 \le i \le m$. Here $s_i(R)$, $1 \le i \le m$, are the sons of the root R arranged in some specific order, and the value of m depends on the degree of the node R.

The postorder traversal of (R,T) is a systematic visit to the nodes of T using the following algorithm:

```
recursive procedure POSTORDER(R,T);
begin
   comment Let (s_i(R),T_i(R)), i=1,...,m be the rooted
```

```
            subtrees under R in (R,T);
      for i:= 1 step 1 until m do
         POSTORDER(s_i(R),T_i(R));
      Visit node R;
   end.
```

If we number the nodes of the tree in the same order as the sequence of node "visits", we obtain an ordering which generally yields a small envelope. In figure 4.3 [Knuth68,p.363], if we choose node I to be the root and the arrangements of the subtree as shown, the postordering α on T will then be:

J, N, L, K, E, F, A, B, D, C, G, M, H, I.

Here the set of potential fill is:

$$Pfill(T_\alpha) = \{ \{G,A\}, \{G,B\}, \{G,D\} \};$$

and they are denoted by dotted lines in the figure.

Evidently, the envelope size depends on the order of the subtrees $(s_i(R),T_i(R))$, $1 \le i \le m$. To see how this arrangement will affect the envelope size, we proceed to characterize $Pfill(T_\alpha)$, where α is the postordering on (R,T) with a <u>given subtree arrangement</u>. The following lemma follows directly from the ordering algorithm.

<u>Lemma 4.1</u>: Let u,v be nodes in the α-labelled rooted tree (R,T). Then $\alpha^{-1}(u) < \alpha^{-1}(v)$ if and only if

either $u \in T_k(v)$ for some k,

or $u \in T_j(z)$ and $v \in T_i(z)$ where $j<i$ for some node z. □

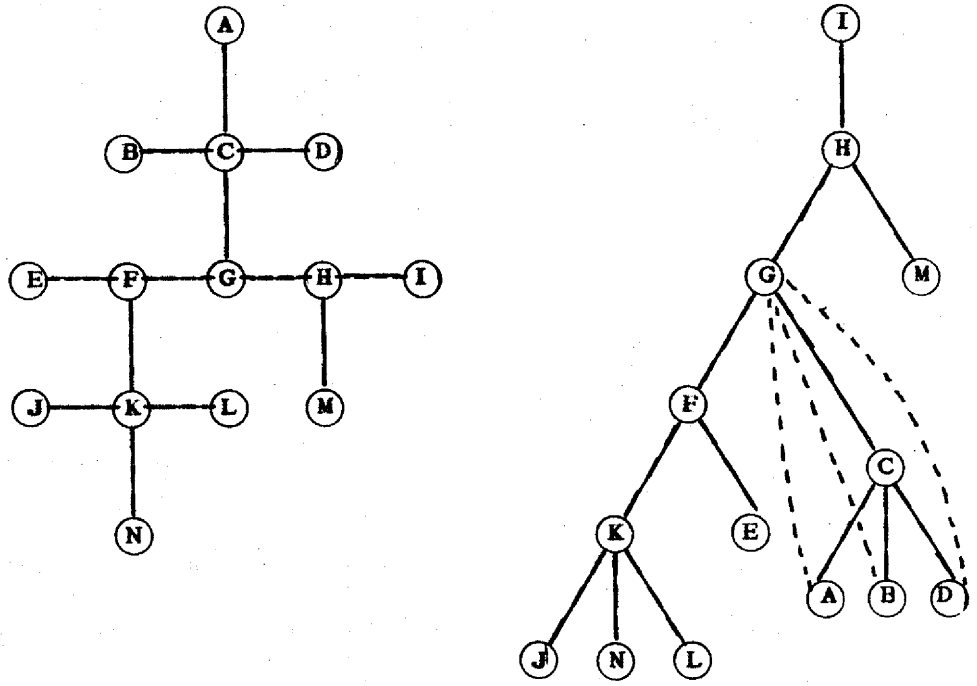


Figure 4.3: A postordering on a rooted tree

Lemma 4.2: Let $\alpha^{-1}(w) < \alpha^{-1}(u) < \alpha^{-1}(v)$. If w is a descendant of v, so is the node u.

Proof: Assume for contradiction that u is not a descendant of the node v. By lemma 4.1, $\alpha^{-1}(u) < \alpha^{-1}(v)$ implies the existence of some node z such that $u \in T_j(z)$ and $v \in T_i(z)$, where $j<i$. Since w is a descendant of v, $w \in T_i(z)$. Yet, this would imply, again by lemma 4.1, that $\alpha^{-1}(u) < \alpha^{-1}(w)$. □

A node s is said to be a **younger son** of x if $s = s_i(x)$ for some $2 \leq i \leq m$.

Theorem 4.4: Let $\alpha^{-1}(u) < \alpha^{-1}(v)$. Then $\{u,v\} \in Pfill(T_\alpha)$ if and only if u is a descendant of v's younger sons.

Proof: "if part" Let $T_1(v), T_2(v), .., T_m(v)$ be the subtrees under v. Consider

$$u \in \bigcup \{T_i(v): 2 \le i \le m\} \setminus Adj(v).$$

Since $Adj(T_1(v)) = \{v\}$ and $\alpha^{-1}(u) < \alpha^{-1}(v)$, we have

$$v \in Adj(\{\alpha(1),\ldots,u\}) \setminus Adj(u).$$

So, by definition, $\{u,v\} \in Pfill(T_\alpha)$.

"only if part" Assume that $\{u,v\} \in Pfill(T_\alpha)$. Now that $v \in Adj(\{\alpha(1),\ldots,u\}) \setminus Adj(u)$, there exists some $k < \alpha^{-1}(u)$ so that $v \in Adj(\alpha(k))$. Thus, $\alpha(k)$ is a son of v. Together with the relation $k < \alpha^{-1}(u) < \alpha^{-1}(v)$, we conclude by lemma 4.2 that $u \in T_i(v)$. Finally, i cannot be one because $\alpha(k)$ is numbered before u, and $\alpha(k)$ is a son of v. □

For a rooted tree (R,T) with $T_i(R)$, $1 \le i \le m$ as subtrees under R, it follows from theorem 4.4 that the number of potential fills due to the node R is given by:

$$\sum_{i=2}^{m} |T_i(R)| - m + 1.$$

In the context of profile minimization, the best way to arrange the subtrees becomes immediate: always pick the one with the greatest number of nodes as the first subtree. This will ensure a minimal set of potential-fill in using the postorder algorithm.

To make the selection possible, we have to know, for node x in the rooted tree (R,T), the number D(x) of descendants x has in (R,T). Knuth [Knuth68] introduces a **locally-defined function** in a tree as a function of the nodes such that the functional value at a node depends only on the node and the functional values of its sons. It is evident that D(x) is a locally-defined function:

$$D(x) = \Sigma \{D(s_i): 1 \leq i \leq m\} + m,$$

where s_i, $i=1,\ldots,m$ are the sons of x in T. Locally-defined functions can be computed very easily, provided that the data representation allows an efficient retrieval of adjacent sets (e.g. adjacency structure [Tarjan74]). If there are N nodes in the rooted tree, D(x) can be determined in O(N) edge inspections.

The algorithm POSTORDER is only applicable to rooted trees. Thus we are left with the problem of finding an appropriate root for a given tree to start the algorithm. An apparently good one is a **peripheral node**. This choice is crucial in the new algorithm described in the next section. To determine a peripheral node for trees requires at most O(N) edge inspections and we will discuss this in more detail in section 4.4.

4.3 A Minimal Envelope Ordering for Trees

The postorder scheme in section 4.2 is a simple, efficient and yet quite effective algorithm for reducing profiles of tree structure. Unfortunately, it does not guarantee a minimal envelope. In figure 4.4, α is a postordering, but we can find another ordering $\bar{\alpha}$ such that

$$Pfill(T_{\bar{\alpha}}) \subsetneq Pfill(T_{\alpha}).$$

In this section, we will modify the postorder scheme to a new algorithm that will always generate a minimal envelope ordering. Let us first study the above example more carefully and relate it to the result of theorem 3.3. The node $\alpha(12)$ is a cutnode in the tree and also in the extension graph $(X,Env(T_{\alpha}))$. There are three components with respect to this separator $\{\alpha(12)\}$:

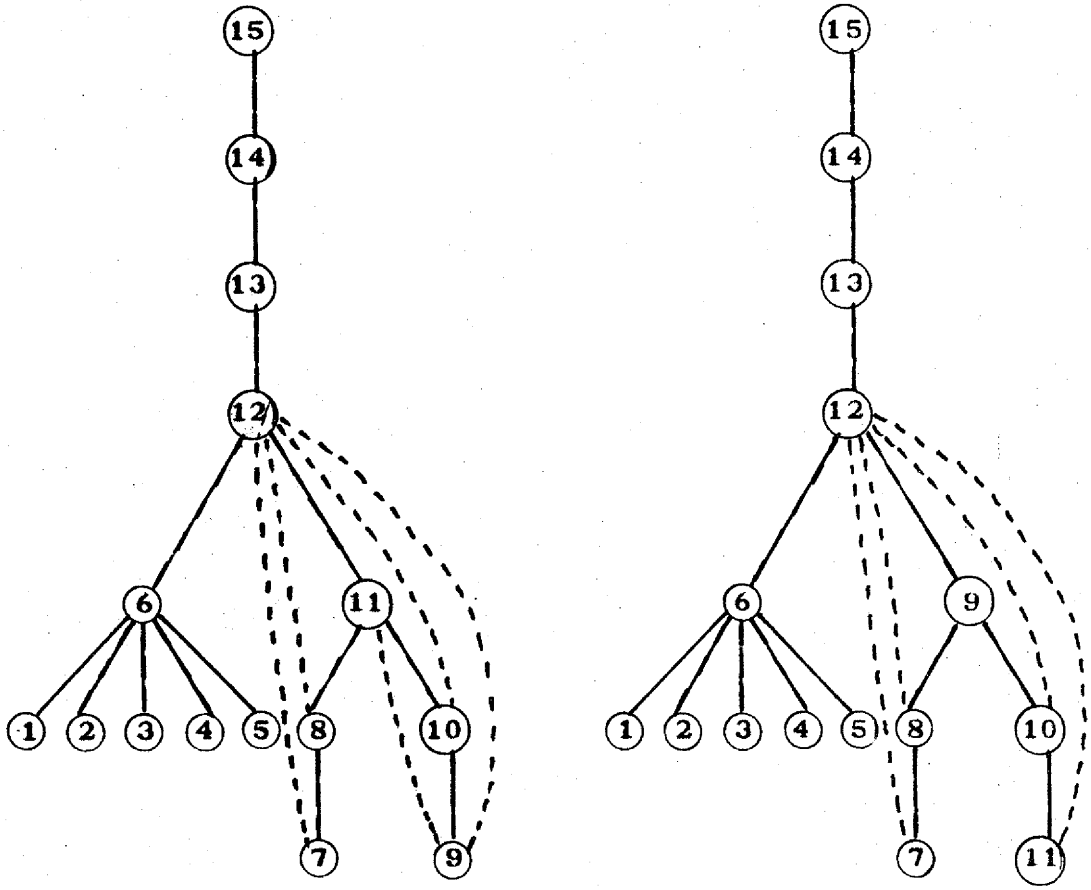
$$C_1 = \{\alpha(1),\alpha(2),\alpha(3),\alpha(4),\alpha(5),\alpha(6)\},$$

$$C_2 = \{\alpha(7),\alpha(8),\alpha(9),\alpha(10),\alpha(11)\},$$

and $$C_3 = \{\alpha(13),\alpha(14),\alpha(15)\}.$$

Theorem 3.3 says that we cannot remove any dotted edges $\{\alpha(12),x\}$, where $x \in C_2$, in the set of potential fills.

But if we consider the component C_2, it is clear that C_2 has a full envelope ordering so that $\{\alpha(12),\alpha(9)\}$ in $Pfill(T_{\alpha})$ can be removed. This suggests that in the postorder scheme for (R,T) with subtrees $T_1(R),\ldots,T_m(R)$ under R, we should treat $T_2(R),\ldots,T_m(R)$ as general subtrees rather than as rooted subtrees. In the following,



A postordering α A better ordering $\bar{\alpha}$

Figure 4.4: Postordering α is not minimal

we describe our modified algorithm MET (Miminal Envelope ordering for Trees) using two recursive procedures.

```
recursive procedure MET(T);
begin
   comment MET generates a minimal envelope ordering
           for a given tree T;
```

```
recursive procedure PORDER(R,T);
   comment PORDER numbers the tree T rooted at R in
           a postorder-like sequence.  T_i(R), 1≤i≤m,
           are the subtrees under the root R;
   begin
      if m > 0 then
         begin
         comment Let s_1(R) be the first son of R;
            PORDER(s_1(R),T_1(R));
            for i:= 2 step 1 until m do
               MET(T_i(R))
         end;
      Number R
   end PORDER;
   R:= a peripheral node of T;
   PORDER(R,T)
end.
```

For a given tree, there is a class of labellings that can be produced by the algorithm MET, depending on the starting node and the subtree arrangements. The better labelling in figure 4.3 belongs to this class. For our discussion, we denote $\bar{\alpha}$ an ordering obtained by MET. We prove through a sequence of lemmas that $\bar{\alpha}$ is a minimal envelope ordering.

Lemma 4.3: If R is a peripheral node in a tree, then deg(R) = 1. □

Consider the tree T=(X,E) rooted at R. Let $\hat{R}$ be the **first** descendant of R that has more than one son, and let $T_i(\hat{R})$ $1 \le i \le m$ be the subtrees under $\hat{R}$. By the definition of $\hat{R}$, $m \ge 2$. Here the first subtree is assumed to have the greatest number of nodes, i.e. $|T_1(\hat{R})| \ge |T_i(\hat{R})|$ for $i=1,\ldots,m$. Note that $\hat{R} \neq R$ because of lemma 4.3.

Lemma 4.4: If $\hat{R} \in Adj(R)$, then $|T_i(\hat{R})| = 1$ for $i=2,..,m$.

Proof: If for some $i \ge 2$, $|T_i(\hat{R})| \ge 2$, R cannot be a peripheral node. □

Lemma 4.5: $\{x,\hat{R}\} \in Pfill(T_{\bar{\alpha}})$ if and only if x is a descendant of $\hat{R}$'s younger sons.

Proof: The proof is the same as theorem 4.4. □

Lemma 4.6: $\hat{R}$ is a cutnode of the extension graph $(X,Env(T_{\bar{\alpha}}))$ of T.

Proof: It is clear that the removal of $\hat{R}$ disconnects the extension graph into m+1 components:

$$T_1(\hat{R}),\ T_2(\hat{R}),\ \ldots,\ T_m(\hat{R}),\ Ancestor(\hat{R}),$$

where $Ancestor(\hat{R})$ is the set of ancestor of $\hat{R}$ in the rooted tree (R,T). □

Lemma 4.7: Let $\{\hat{R},y\} \in Pfill(T_{\bar{\alpha}})$. Then the graph $(X,Env(T_{\bar{\alpha}})\setminus\{\hat{R},y\})$ does not have full envelope orderings.

Proof: By lemma 4.5, $y \in T_i(\hat{R}) \setminus Adj(\hat{R})$ for some $i > 1$. Thus $|T_i(\hat{R})| > 1$, so that $|T_1(\hat{R})| > 1$ and by lemma 4.4, $R \notin Adj(\hat{R})$.

This means that $\hat{R}$ is a cutnode of the extension graph with components $T_1(\hat{R}),..,T_m(\hat{R})$, $Ancestor(\hat{R})$, where

$$T_1(\hat{R}) \not\subset Adj(\hat{R}) \text{ and } Ancestor(\hat{R}) \not\subset Adj(\hat{R}).$$

Hence, by theorem 3.3, the removal of some $\{\hat{R},y\}$ from the extension graph $(X,Env(T_{\bar{\alpha}}))$ gives a graph with no full envelope ordering. □

Lemma 4.8: Let $s_1(\hat{R})$ be the first son of $\hat{R}$. If $\{s_1(\hat{R}),y\} \in Pfill(T_{\bar{\alpha}})$, then $(X,Env(T_{\bar{\alpha}})\setminus\{s_1(\hat{R}),y\})$ does not have a full envelope ordering.

Proof: The node $s = s_1(\hat{R})$ is also a cutnode of the extension graph $(X,Env(T_{\bar{\alpha}})$ and the corresponding components are: $T_1(s), \ldots, T_m(s), X \setminus T_1(\hat{R})$.
The result then follows from the same argument as in lemma 4.7. □

Theorem 4.5: $\bar{\alpha}$ is a minimal envelope labelling.

Proof: In view of the recursive use of the algorithm, the theorem follows from a recursive use of the results in lemmas 4.7 and 4.8. □

We point out here that in general $\bar{\alpha}$ is not necessarily a **minimum** envelope labelling. The tree in example 4.4 has a better ordering as shown in figure 4.5.

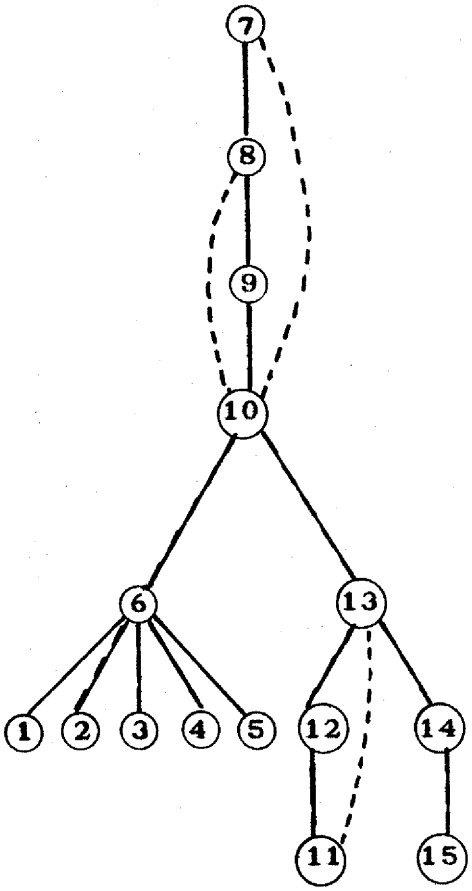


Figure 4.5: A better ordering than $\bar{\alpha}$ in figure 4.4

Although $\bar{\alpha}$ may not be a minimum ordering, the amount of potential fill in $\bar{\alpha}$ is reasonably small. A bound is established in the following theorem.

Theorem 4.6: $|Pfill(T_{\bar{\alpha}})| \leq N \log_2 N$.

Proof: Let $p(N)$ be the maximum possible number of potential fills on applying PORDER in the algorithm MET to a rooted tree with N nodes. We shall use induction to show $p(N) \leq N \log_2 N$.

Clearly, $p(2) \leq 2$. Assume the inequality holds for all $k<N$. Let (R,T) be a rooted tree with N nodes which suffers $p(N)$ number of potential fills. Let $T_1(R)$, $T_2(R)$, ..., $T_m(R)$ be the subtrees under the root R where $|T_1(R)| \geq |T_i(R)|$. Using lemma 4.5 and the inductive assumption, we have

$$p(N) \leq \sum_{i=1}^{m} p(|T_i(R)|) + \sum_{i=2}^{m} |T_i(R)| - m + 1$$

$$\leq \sum_{i=1}^{m} |T_i(R)| \log_2|T_i(R)| + \sum_{i=2}^{m} |T_i(R)|$$

$$\leq |T_1(R)| \log_2|T_1(R)| + \sum_{i=2}^{m} |T_i(R)|(\log_2|T_i(R)|+1).$$

But for $i=2,\ldots,m$, $|T_i(R)| \leq (N-1)/2$ so that $\log_2|T_i(R)| \leq \log_2 N - 1$. Thus,

$$p(N) \leq |T_1(R)| \log_2|T_1(R)| + \sum_{i=2}^{m} |T_i(R)| \log_2 N$$

$$\leq N \log_2 N.$$

The theorem then follows from $|Pfill(T_{\bar{\alpha}})| \leq p(N)$. □

4.4 Implementation and Time Complexity of MET

A peripheral node of a tree or subtree is required recursively in the algorithm MET. We first consider how such nodes can be determined efficiently using rooted level structures. The following lemma provides an almost trivial way to determine a peripheral node of a tree.

Lemma 4.9: Let x be a node in a tree T, and its corresponding rooted level structure be

$$\mathcal{L}(x) = \{L_0(x), L_1(x), \ldots, L_{l(x)}(x)\}.$$

Then any y in $L_{l(x)}(x)$ is a peripheral node.

Proof: The lemma follows from the fact that any two nodes in a tree are connected by exactly one path. □

After finding a peripheral node, say R, we have to compute the sizes of the subtrees, i.e. the number of descendants D(x) under each node x in the rooted tree (R,T). The quantity D(x) can be determined quite simply by running through the rooted level structure at R bottom-up once:

```
for x ∈ X do
    D(x):= 1;

for i:= l(x) step -1 until 1 do
begin
    for y ∈ L_i(R) do
    begin
```

```
        for z ∈ Adj(y) ∩ L_{i-1}(R) do
            D(z):= D(z) + D(y);
        end;
    end;
```

It is evident that a peripheral node and the corresponding function D(x) can be determined in kN edge inspections. Here k is a constant. We are now ready to determine the asymptotic time complexity of the algorithm MET. Define c(N) to be the maximum possible cost required by MET to find a minimal envelope labelling for a tree of N nodes. We measure the cost by the number of edge inspections.

Theorem 4.7: $c(N) \leq k\, N \log_2 N$.

Proof: We shall use induction to show the inequality. Clearly $c(2) \leq 2k$. Assume the result holds for all $k<N$. Let T be a tree with N nodes that requires c(N) number of edge inspections to find a minimal envelope labelling.

Let R be a peripheral node found by MET and $T_1(R)$, $T_2(R)$, ..., $T_m(R)$ be the subtrees under R. From the algorithm and our induction assumption, we have

$$c(N) \leq kN + c(|T_1(R)|) - k|T_1(R)| + \sum_{i=2}^{m} c(|T_i(R)|)$$

$$\leq k(N-|T_1(R)|) + \sum_{i=1}^{m} k|T_i(R)| \log_2|T_i(R)|$$

$$= k + k|T_1(R)| \log_2|T_1(R)| + \sum_{i=2}^{m} k|T_i(R)|(\log_2|T_i(R)|+1).$$

Since $|T_i(R)| \leq (N-1)/2$, for $i=2,..,m$,

$$c(N) \leq k + k \log_2 N \sum_{i=1}^{m} |T_i(R)|$$

$$\leq k \log_2 N(1+ \sum_{i=1}^{m} |T_i(R)|)$$

$$= k N \log_2 N.$$ □

4.5 Experimental Results

The algorithm MET has been implemented in ALGOL W [Bauer69] and run on an IBM 360/75. The program is applied to a sequence of randomly generated trees of different sizes in order to find the experimental running time.

The test trees are generated recursively as follows. Let T_{N-1} be a random tree with N-1 nodes. A node x_i is selected at random from T_{N-1}. We then add the new node x_N and the edge $\{x_N, x_i\}$ to form T_N.

The test results are tabulated in table 4.1. For each N, the result is the average of twenty random trees. The plots in figures 4.6 and 4.7 show that the cost is indeed proportional to $N \log_2 N$. By approximating the data with a straight line (in the least square sense), we obtain

$$c(N) \simeq 1.5\ N \log_2 N,$$

and $$time(N) \simeq 0.0002\ N \log_2 N \text{ seconds.}$$

We also note that the number of potential fills is bounded by $N\log_2 N$; in most cases Pfill is much smaller.

N	N $\log_2 N$	$\lvert Pfill(T_{\overline{a}}) \rvert$	No of edge inspection	Time in seconds
100	664.4	76.9	1220.6	0.14
200	1528.8	195.8	2621.5	0.28
300	2468.6	346.0	4147.1	0.46
400	3457.5	504.9	5710.3	0.62
500	4482.9	682.1	7336.1	0.85
600	5537.3	838.2	8889.1	1.02
700	6615.9	1041.9	10627.6	1.22
800	7715.1	1203.7	12202.3	1.42
900	8832.4	1386.3	13863.7	1.68
1000	9965.8	1642.2	15801.2	1.91
1100	11113.6	1796.4	17344.7	2.13
1200	12274.6	2036.7	19227.5	2.39
1300	13447.6	2189.8	20769.1	2.69
1400	14631.7	2438.9	22688.5	2.91
1500	15826.1	2595.0	24238.3	3.17

Table 4.1: Average results of MET on randomly generated trees.

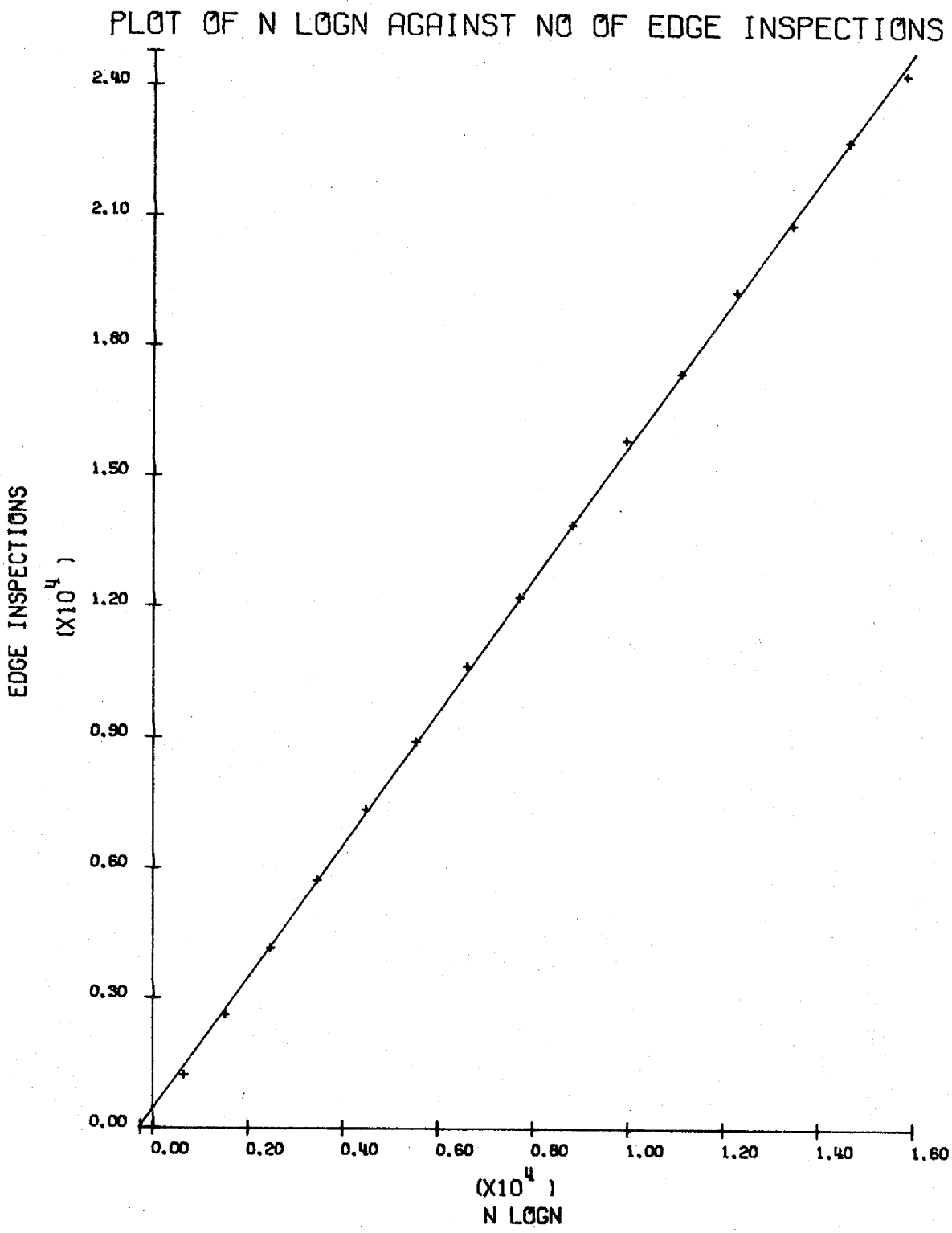


Figure 4.6: Plot of $N \log_2 N$ against the number of edge inspections.

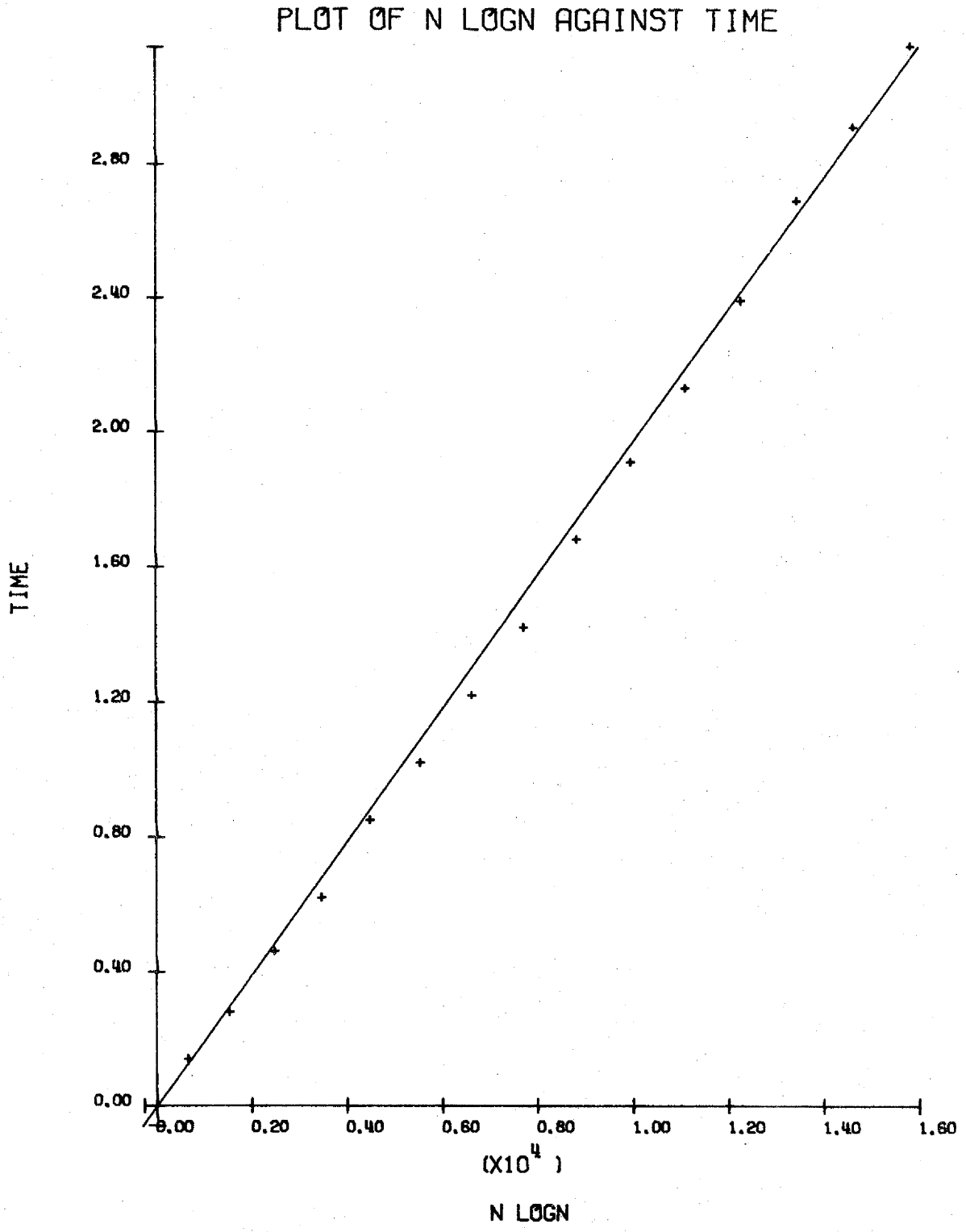


Figure 4.7: Plot of N $\log_2$N against time

hapter 5 On Reducing the Profile of General Graphs

In this chapter, we will analyse, test and compare several popular profile reduction schemes for general graphs. They include the algorithms of Cuthill-McKee, George (reverse Cuthill-McKee), King and Levy. These algorithms are tested on the Cuthill and Everstine's collection of sparse matrices derived from structural engineering problems. The performance of the recent algorithm by Gibbs, Poole and Stockmeyer, which does an excellent job on bandwidth reduction, is also given for comparison.

5.1 Introduction

From the way sparsity is exploited in profile methods, it is ideal to have minimum envelope orderings (3.18). However, such orderings seem to be difficult to obtain. An efficient algorithm for finding a minimal envelope ordering for general graphs has not yet been found. In practice, it is relatively unimportant whether the minimum profile is achieved or a minimal envelope is obtained. We are more interested in finding reordering algorithms which produce a profile reasonably close to minimum in an economical amount of time.

Reordering algorithms for reducing profiles may be classified as direct or iterative. Direct schemes usually begin with a starting node (or a set of starting nodes) and

construct the node ordering according to some strategy based on the structure of the graph. Usually, one pass through the graph structure is enough to complete the labelling. On the other hand, iterative schemes begin with an initial ordering and attempt to improve it by making appropriate node-label interchanges.

The existing iterative methods ([Akyuz68], [Alway65]) have a common shortcoming: too expensive in terms of computer time. They are typically $O(N^3)$ or $O(N^4)$ algorithms. A comparison with the operation count $\frac{1}{6}N^3+O(N^2)$ in (1.4) for full symmetric factorization shows that they can be of little practical value. For more discussions on iterative ordering algorithms, see [Bolstad73] and [George71,p106].

In this chapter, we study direct reordering schemes for minimizing profiles. Two approaches can be identified in the current effective direct algorithms: one makes use of the equation

$$|Env(G)| = \Sigma\, w_i(G),$$

and the other works on the identity

$$|Env(G)| = \Sigma\, \beta_i(G).$$

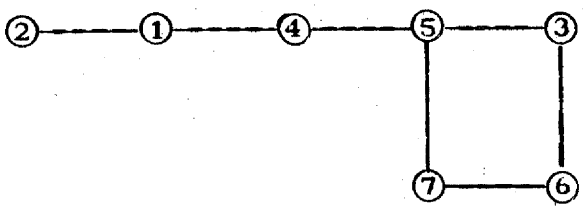
These are local algorithms, in the sense that a local quantity β_i or w_i is minimized so as to reduce the value of the global quantity $|Env(G)|$. The King [King70] and Levy [Levy71] algorithms belong to the former class; while the Cuthill-McKee [Cuthill69] and reverse Cuthill-McKee schemes [George71] to the latter. We shall analyse their performances and give some experimental comparisons of these algorithms.

5.2 Data Structures for Graphs

It is well known that the performances of graph algorithms are in general sensitive to the way the graphs are represented. In this section, we consider the data structures for graphs in the effective implementation of graph ordering algorithms. Since the basic operation in most direct schemes is that of retrieving adjacent relations between nodes, we need a representation which facilitates the retrieving operation and which is economical in storage.

A survey of current representations of general sparse matrices (and hence sparse labelled graphs) is given by Pooch and Nieder [Pooch73]. It should be noted that graphs are always represented as labelled ones. For our purpose, the most appropriate data scheme is a list structure representation. Let $G=(X,E)$, where $|X|=N$. Following [Tarjan72], we define an adjacency list for a node $x \in X$ to be a list containing all the nodes in $Adj(x)$. An adjacency structure for G is the set of adjacency lists for all the nodes in G. Clearly, an adjacency structure uses an amount of storage linear in $|X|$ and $|E|$.

An adjacency structure can be implemented most economically by storing the adjacency lists in a linear array sequentially [Eisenstat74]. The main storage is a vector ALIST of length $2|E|$ for the set of adjacency lists, while an auxiliary array XLIST is kept to store the



ALIST	2 4	1	5 6	1 5	3 4 5	3 7	5 6

XLIST	1	3	4	6	8	11	13	15

Figure 5.1 Adjacency sequential lists of a labelled graph

beginning addresses of the adjacency lists. The scheme is illustrated in figure 5.1.

The total storage requirement is $2|E|+|X|$ locations. It is often convenient to let XLIST(N+1) point to the next availabe storage location in the main array ALIST. The neighbors for node i can then be retrieved as:

$$\{ALIST(j): XLIST(i) \leq j < XLIST(i+1)\}.$$

Another common form of adjacency structure representation is the connection table ([Crane75], [Wang73]). The table has N rows and m columns where

$$m = \max \{deg(x): x \in X\}.$$

The adjacency list for node i is stored in row i of the connection table. The table corresponding to the labelled graph in figure 5.1 is given in figure 5.2.

$$\begin{bmatrix} 2 & 4 & 0 \\ 1 & 0 & 0 \\ 5 & 6 & 0 \\ 1 & 5 & 0 \\ 3 & 4 & 7 \\ 3 & 7 & 0 \\ 5 & 6 & 0 \end{bmatrix}$$

Figure 5.2 Connection table for graph in figure 5.1

Any variant of the above schemes suffers one disadvantage. Unless the degrees of the nodes are known beforehand, it is difficult to construct the storage vector when the graph is given as a list of edges. In that case, a link field is necessary, so that $4|E|+|X|$ storage locations are required. Figure 5.3 is an example of a computer implementation of adjacency linked lists [Rheinboldt73]. Three arrays HEAD(*), NBRS(*), and LINK(*) are used. HEAD(i) starts the adjacency list for node i. NBRS contains

	HEAD
1	13
2	14
3	9
4	8
5	11
6	6
7	12

	NBRS	LINK
1	7	-6
2	6	-7
3	5	-4
4	4	-5
5	6	-3
6	3	1
7	4	-1
8	1	3
9	5	5
10	3	4
11	7	10
12	5	2
13	2	7
14	1	-2

Figure 5.3 Adjacency linked lists for graph in figure 5.1

the neighbor while LINK stores the pointer to the next neighbor of the node under consideration. If the quantity of LINK is negative, in fact -i, we have come to the end of the list for node i.

In this setting where {NBRS(j),NBRS(j+1)} $\in$ E for odd j, the graph may be represented without the array NBRS. Of course, additional work has to be performed in searching through the links for the indices of neighbors. For details, see [Rheinboldt73].

The different implementations of the adjacency structure require roughly the same amount of time in retrieving the adjacent set of a node. They differ in the amount of storage and the degree of flexibility. We have described them in fair detail, and in chapter 6 they will be compared with some different representation of a special class of graphs.

5.3 Algorithms Minimizing the i-th Frontwidth

5.3.1 Levy Algorithm

The algorithm by Levy [Levy71] is designed to produce an ordering that will yield a small envelope. It may be described as follows.

```
procedure LEVY( X, E, α);
begin
   comment (X,E) is the input graph and α is the
                output Levy labelling;

   i:= 0;
   while i < |X| do
   begin
      x:= an unlabelled node;
      for unlabelled y ∈ X do
      begin
         if |Adj( {α(1),..,α(i),y} )| < |Adj( {α(1),..,α(i),x} )|
            then x:= y
      end;

      i:= i+1;
      α(i):= x
   end
end.
```

The algorithm has N steps; at the i-th of which node x is labelled i if it increases the frontwidth least among the set of unlabelled nodes. Although a minimum w_i results at each stage of the labelling, the algorithm may not produce an ordering with a global minimum profile. This is illustrated by the example [Rose72a] in figure 5.4.

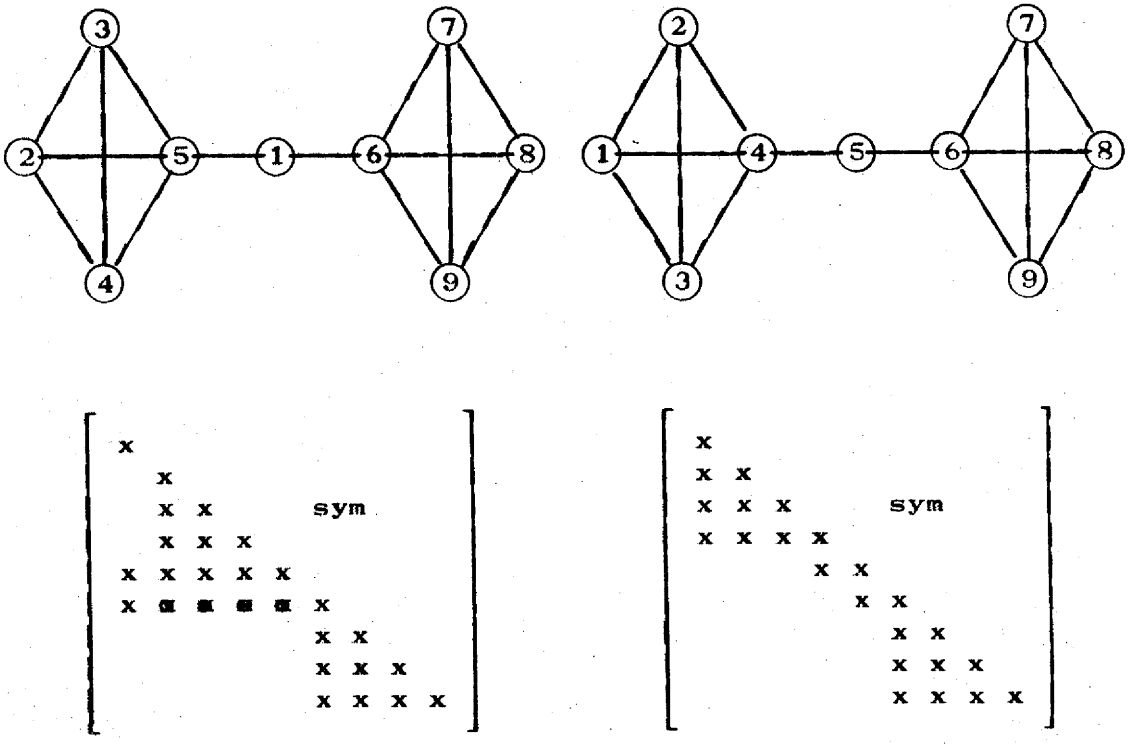


Figure 5.4 A Levy ordering which is not minimum

The main shortcoming of the Levy algorithm is its inefficiency. At step i, it performs a time-consuming search among the set of (N-i+1) currently unlabelled nodes. It is of $O(N^2)$ complexity. The overall efficiency of the Levy algorithm can be improved by using an observation in lemma 3.4:

$$w_i(G) \leq w_{i+1}(G) + 1.$$

So, if $|Adj(\{\alpha(1),..,\alpha(i),x\})| = |Adj(\{\alpha(1),..,\alpha(i)\})| - 1$, we know that it is unnecessary to test the remaining unnumbered nodes. Experiments show that this additional check quite often reduces the ordering time.

We include this algorithm in our comparative study primarily because it differs from the rest in being self-starting. The first node labelled is not crucial to the entire ordering, though to some extent it does affect the envelope size (see figure 5.4).

.3.2 King Algorithm

At a step of the Levy algorithm, all the remaining unlabelled nodes are examined. It is possible to improve the efficiency by looking at only those nodes adjacent to the ones already numbered. This is the essential idea in the King algorithm [King70], which has become one of the popular profile reduction schemes. The following is a description of the King algorithm when applied to a connected graph with a given starting node s.

```
procedure KING(X, E, s, α);
begin
   comment (X,E) is the input graph, s the given starting
                 node, and α is the output King labelling;
   i:= 1;
   α(i):= s;
   while i < |X| do
   begin
      x:= a node in Adj({α(1),..,α(i)});
      for y ∈ Adj({α(1),..,α(i)}) do
      begin
         if |Adj({α(1),..,α(i),y})| < |Adj({α(1),..,α(i),x})|
            then  x:= y
      end;

      i:= i+1;
      α(i):= x
   end
end.
```

In case there is more than one node in the i-th front $Adj(\{\alpha(1),..,\alpha(i)\})$ with minimum frontwidth increase, it is a common practice to break the tie by choosing a node whose first neighbor in $\{\alpha(1),..,\alpha(i)\}$ has the smallest subscript. Modification of the above algorithm for disconnected graphs is straightforward. In view of lemma 3.5, a component of the disconnected graph can be detected whenever $Adj(\{\alpha(1),..,\alpha(i)\}) = \emptyset$ for some $i<N$.

The King algorithm uses the same basic idea as the Levy algorithm: it tries to minimize the sum

$$\Sigma\ |Adj(\{\alpha(1),..,\alpha(i)\})|$$

on a local basis. Note that the selection among the nodes in the front can be implemented using the following equivalent condition.

Lemma 5.1: Let x be a node in the i-th front $Adj(\{\alpha(1),..,\alpha(i)\})$. Among the nodes in the front, $|Adj(\{\alpha(1),..,\alpha(i),x\})|$ is minimum if and only if x has the least number of connections to unlabelled and non-frontal nodes.

Proof: The lemma follows from the relation

$$Adj(\{\alpha(1),..\alpha(i),x\}) \cup \{x\}$$
$$= Adj(\{\alpha(1),..,\alpha(i)\}) \cup Adj(x) \cap [X \setminus \overline{Adj}(\{\alpha(1),..,\alpha(i)\})],$$

where $\overline{Adj}(Y) = Adj(Y) \cup Y$ for any subset Y of X. □

Let us denote by G_k the graph labelled using a King ordering. The following theorem provides an interesting property of G_k.

Theorem 5.1: The labelled graph G_k suffers maximal envelope fill.

Proof: Assume that the graph G_k is connected. Since $\alpha(i) \in Adj(\{\alpha(1),..,\alpha(i-1)\})$, for $i>1$, the section graphs $\{\alpha(1),..,\alpha(i)\}$ are connected. By theorem 3.4, G_k suffers maximal fill. The proof can be extended to disconnected graphs, since the King algorithm orders component after component. □

Theorem 5.2: Using the adjacency structure representation, the King algorithm requires $O(|Env(G_k)|)$ comparisons.

Proof: From lemma 5.1, selection from the i-th front can be performed by having for each node, its current number of unlabelled and nonfrontal neighbors. Then, if one sweep through the adjacency structure requires $2|E|$ operations, the King algorithm can be implemented so that it requires

$$4|E| + |Env(G_k)| \text{ comparisons.}$$ □

Like most direct ordering schemes, the King algorithm depends greatly on the starting node. We shall discuss this problem in section 5.5.

.4 Algorithms Minimizing the i-th Bandwidth

.4.1 Cuthill-McKee Algorithm

The Cuthill-McKee algorithm [Cuthill69] is primarily designed to reduce the bandwidth of a sparse graph (or matrix). It is probably the most widely used reduction algorithm. We describe the algorithm for a connected graph with a given starting node as follows.

```
procedure CM(X, E, s, α);
begin
   comment (X,E) is the input connected graph, s the
                starting node, and α will be the output
                Cuthill-McKee ordering;
   i:= 1;
   α(1):= s;
   j:= 0;
   while i < |X| do
   begin
      j:= j+1;
      for x ∈ Adj(α(j)) \ {α(1),..,α(i)}
          in increasing order of degree do
      begin
         i:= i + 1;
         α(i):= x
      end
   end
end.
```

As remarked in [Cuthill69], the Cuthill-McKee numbering scheme corresponds to the breadth-first generation of a spanning tree† of the connected graph G in a level-by-level fashion. In case when G is disconnected, a component of G can be identified whenever the pointer value of j is the same as that of i. The process can be continued by selecting a new and unlabelled node to start the scheme for a new component.

Let y be a labelled node. Consider an unlabelled neighbor x of y. To minimize the bandwidth associated with x (i.e β_i if the node x is numbered i), it is apparent that x should be ordered after y as soon as possible. The Cuthill-McKee algorithm makes use of this observation, and thus may be regarded as a method that reduces the profile of the graph via a local minimization of β_i.

We now analyze some properties of the Cuthill-McKee ordering. Let G_c be the graph labelled by the scheme.

Lemma 5.2: The labelled graph G_c satisfies the monotone profile property (3.19).

Proof: Assume that $f_j(G_c) < f_i(G_c)$. When the unlabelled neighbors of $\alpha(f_j(G_c))$ are numbered, $\alpha(j)$ is labelled, so that $\alpha(j)$ should be numbered before $\alpha(i)$. Thus, $j < i$. □

---

† A spanning tree of a connected graph G is a subgraph of G, which is a tree and which contains all the nodes in G.

Theorem 5.3: The labelled graph G_c suffers maximal envelope fill.

Proof: It follows from lemma 5.2 and theorem 3.5. □

Theorem 5.4: $\theta_E(G_c) = \frac{1}{2} \sum \beta_i(G_c)\ \{\beta_i(G_c)+3\}$. □

An exact complexity bound for the Cuthill-McKee scheme is difficult, due to the problem in estimating the complexity in the sorting step. A rough bound is given below.

Theorem 5.5: If linear insertion is used for sorting, the time complexity of the Cuthill-McKee algorithm is bounded by $O(m|E|)$, where m is the maximum degree of the nodes.

Proof: Assume that linear insertion on t elements requires $\frac{1}{4}t^2$ operations. Then the overall time spent in sorting is less than

$$\begin{aligned} &\tfrac{1}{4} \sum \{\deg(x)^2 : x \in X\} \\ &\leq \tfrac{1}{4}\, m \sum \{\deg(x) : x \in X\} \\ &= \tfrac{1}{2}\, m|E|. \end{aligned}$$

If one sweep through the adjacency structure requires $2|E|$ operations, then the scheme requires less than

$$2|E| + \tfrac{1}{2}m|E| \text{ operations.}$$ □

4.4.2 Reverse Cuthill-McKee Algorithm

In his study of profile methods, George [George71] discovered the reverse Cuthill-McKee (RCM) algorithm, which renumbers the CM ordering in the reverse way. This simple modification often turns out to be superior to the original ordering in several aspects, although the bandwidth remains unchanged. The general superior performance of the reverse algorithm has been reported in the thesis of George [George71] and the survey article by Cuthill [Cuthill72]. In this section, we show that in general the reverse scheme is always at least as good, as far as storage and operation counts are concerned. The method used is graph-theoretic; a proof using a matrix approach has been given by Liu and Sherman [Liu75].

The profile-reducing property of the RCM algorithm can be attributed to the fact that it minimizes the quantity β_i locally in this order

$$\beta_N, \beta_{N-1}, \cdots, \beta_1.$$

After the last node y_N (i.e. the first node in CM), we number all those neighbors of y_N immediately so that the value of β_N will be minimum. The same kind of strategy --- the unlabelled neighbors of node y_i are numbered to minimize β_i --- is followed for the rest of the ordering scheme. This may help to explain the general good performance of the RCM scheme in the context of profile minimization.

Let $\{x_1, x_2, \ldots, x_N\}$

and $\{y_1, y_2, \ldots, y_N\}$

be the CM and the RCM orderings respectively. Thus, x_i and y_{N-i+1} represent the same node in the underlying graph. We also denote by G_c and G_r the graphs labelled by CM and RCM algorithms respectively.

Lemma 5.3: $Adj(\{x_1,\ldots,x_N\}) \subset \{x_{f_i},\ldots,x_{i-1}\}$, where $f_i = f_i(G_c)$.

Proof: Assume for contradiction that there exists

$$x \in Adj(\{x_1,\ldots,x_N\}) \setminus \{x_{f_i},\ldots,x_{i-1}\}.$$

Let $x \in Adj(x_k)$ for some $k > i$. This would imply that $f_k(G_c) < f_i(G_c)$ contradicting the monotone profile property of G_c in lemma 5.2. □

Theorem 5.6: $Env(G_r) \subset Env(G_c)$.

Proof: Consider $\{y_i, y_j\} \in Env(G_r)$, where $j < i$. Thus

$$y_i \in Adj(\{y_1,\ldots,y_j\}) = Adj(\{x_N,\ldots,x_{N-j+1}\}) \subset \{x_{f_{N-j+1}}, \ldots, x_{N-j}\}.$$

That is, $x_{N-i+1} \in \{x_{f_{N-j+1}},\ldots,x_{N-j}\}$,

so that $f_{N-j+1} \le N-i+1 < N-j+1$.

By definition (3.11),

$$\{y_i, y_j\} = \{x_{N-i+1}, x_{N-j+1}\} \in Env(G_c). \quad \square$$

Corollary 5.4: $|Env(G_r)| \le |Env(G_c)|$. □

Theorem 5.7: $Fill(G_r) \subset Fill(G_c)$.

Proof: It follows from definition and theorem 5.3,

$$\begin{aligned} Fill(G_r) &\subset Env(G_r) \setminus E \\ &\subset Env(G_c) \setminus E \\ &= Pfill(G_c) \\ &= Fill(G_c). \end{aligned}$$ □

Lemma 5.5: For $1 \le i \le N$,

$$w_{N-i+1}(G_r) \le \beta_i(G_c).$$

Proof: Note that

$$\begin{aligned} w_{N-i+1}(G_r) &= |Adj(\{y_1,\ldots,y_{N-i+1}\})| \\ &= |Adj(\{x_N,\ldots,x_i\})|, \end{aligned}$$

and $$\begin{aligned} \beta_i(G_c) &= i - f_i(G_c) \\ &= |\{x_{f_i(G_c)}, \ldots, x_{i-1}\}|. \end{aligned}$$

The result then follows from lemma 5.3. □

Theorem 5.8: $\theta_E(G_r) \le \theta_E(G_c)$. □

Theorem 5.9: $Env(G_r) = Env(G_c)$ if and only if G_r has the monotone profile property.

Proof: "if part" If G_r has the monontone profile property, the proof in lemma 5.3 and theorem 5.6 applies so that $Env(G_c) \subset Env(G_r)$.

"only if part" If G_r does not have the property, there exist $j < i$ such that $f_i(G_r) < f_j(G_r)$. Then

$$\{y_{f_i(G_r)}, y_j\} \in Env(G_c) \setminus Env(G_r).$$ □

5 Choice of Starting Node --- On Pseudo-Peripheral Nodes

The effectiveness of most direct algorithms depends on the proper choice of the starting node. The treatment of tree structures in chapter 4 shows that it is essential to start with a peripheral node for minimal envelope ordering. In the context of bandwidth minimization, many researchers advocate the use of peripheral nodes ([Cheng73] [Gibbs74a]). The choice is motivated from a bound on the bandwidth of orderings associated with level structures; see [Gibbs74a].

Original versions of the CM (hence RCM) and the King algorithms perform a time-consuming search for a reasonably good starting nodes. Since these algorithms are similar in spirit to direct bandwidth reduction schemes, it is appropriate to start them with a peripheral node.

However, no efficient algorithm exists to determine peripheral nodes. A novel approach proposed by Gibbs, Poole and Stockmeyer is to determine a node x whose eccentricity $l(x)$ is close to the diameter of the graph. They introduce the notion of pseudo-peripheral nodes. A pseudo-peripheral node may be formally defined as a node x that satisfies the condition: for $y \in X$,

$$\text{if } d(x,y) = l(x), \text{ then } l(y) = l(x).$$

Clearly, any peripheral node is also a pseudo-peripheral node.

Lemma 5.7: ([Gibbs74a]) In a tree, pseudo-peripheral nodes are peripheral. □

But lemma 5.7 may not be true for general connected graphs. In the example of figure 5.5, node 1 is a pseudo-peripheral node, but only node 6 and 9 are peripheral.

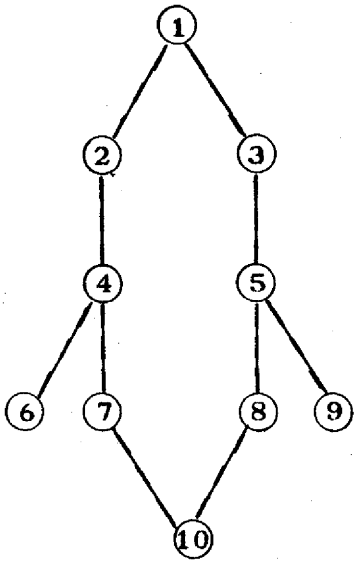


Figure 5.5: A pseudo-peripheral node

Gibbs et al also provide an efficient way of finding a pseudo-peripheral node. Their method is based on the observation that: if y is a node in the last level of the level structure at x, then $l(x) \leq l(y)$. We now describe their algorithm below.

```
procedure PSEUDO_PERIPHERAL(X, E, s);
begin
   comment (X,E) is the input connected graph and s
                 will be the returned pseudo-
peripheral node;
      s:= a node of minimum degree;
      L(s) ⟵ {L0(s), L1(s), ..., Ll(s)(s)};
loop: for x ∈ Ll(s)(s)
          in increasing order of degree do
      begin
         L(x) ⟵ {L0(x), ..., Ll(x)(x)};
         if l(x) > l(s) then
         begin
            s:= x;
            goto loop
         end
      end
   end.
```

In their actual implementation, the execution time is significantly improved by rejecting "wide" rooted level structures. In this way, many such structures $L(x)$ in the loop are discarded while partially formed. For details, see [Crane75].

The incorporation of the idea of pseudo-peripheral nodes to the RCM and the King algorithms proves to be a success. The experimental results in section 5.7 show the effectiveness of such combinations.

5.6 Hypothetical Examples

All the ordering algorithms described are heuristic. In general, they do not produce minimal envelope orderings, let alone minimum. A natural question then arises: as far as the profile is concerned, how bad can these algorithms be? In this section, we give examples where the algorithms produce orderings such that the profile is arbitrarily greater than the minimum profile.

To a certain extent, the Levy algorithm resembles the minimum degree algorithm [Rose72a], which is oriented to fill minimization. The example, used by Rose to show the inadequacy of the minimum degree scheme, serves our purpose. We reproduce it in figure 5.6.

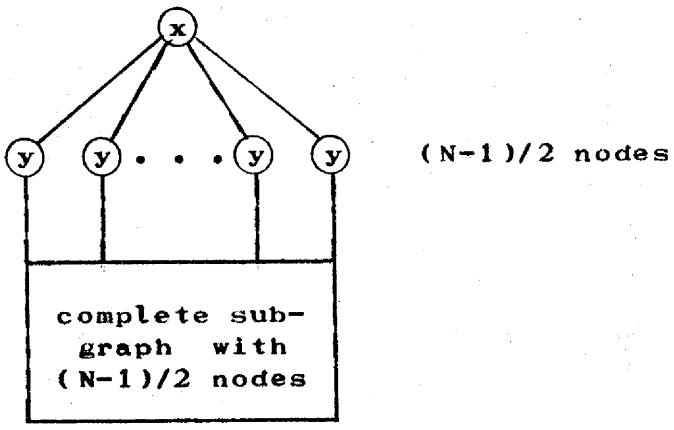


Figure 5.6 Example with poor Levy ordering.

The graph has a complete subgraph of $\frac{1}{2}(N-1)$ nodes, and each of the other $\frac{1}{2}(N-1)$ 'y' nodes in the graph is connected to every node of the subgraph. The Levy algorithm numbers the node 'x' first, so that

$$|Env(G_l)| = \tfrac{1}{2} N^2 + O(N),$$

$$\theta_E(G_l) = \tfrac{1}{6} N^3 + O(N^2).$$

However, for a minimum envelope ordering, for example, one by the RCM scheme, we have

$$|Env(G_r)| = \tfrac{3}{8} N^2 + O(N),$$

$$\theta_E(G_r) = 1\tfrac{1}{2} N^3 + O(N^2).$$

The difference in this example is quite significant. Yet, the RCM and King schemes can even be much worse for some hypothetical examples. Consider the example in figure 5.6. The RCM and King algorithms produce essentially the same node orderings. Generalizing the example to a graph of N nodes, we have

$$|Env(G_k)| = |Env(G_r)| = \tfrac{1}{4} N^2 + O(N),$$

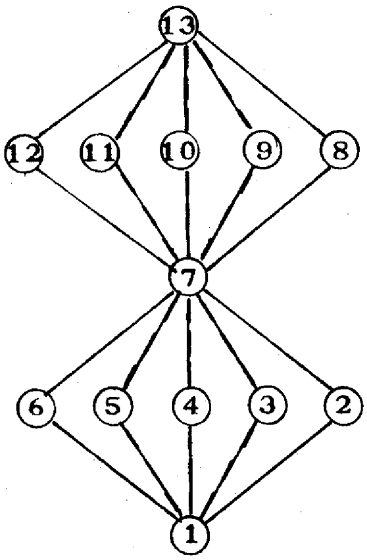
$$\theta_E(G_k) = \theta_E(G_r) = 2\tfrac{1}{4} N^3 + O(N^2).$$

But for a minimum envelope ordering, for example one by the Levy algorithm, we have

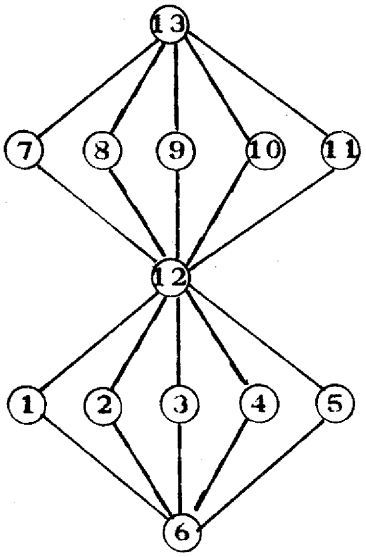
$$|Env(G_l)| = 2 N + O(1),$$

$$\theta_E(G_l) = 5 N + O(1).$$

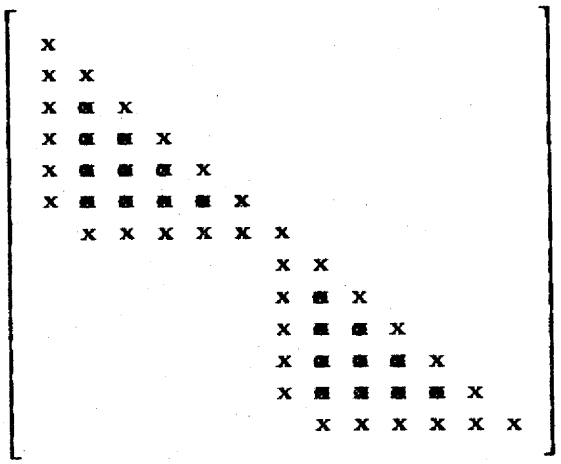
Fortunately, for application problems, these schemes seldom exhibit such erratic behavior; usually they produce profiles satisfactory for practical purposes. In the next section. we will test the behavior of the various algorithms when applied to a collection of application problems.



RCM/King Ordering



Minimum Ordering



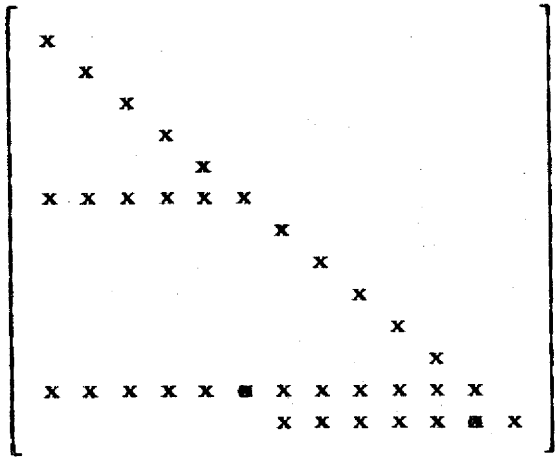


Figure 5.6 Example with poor RCM/King ordering.

5.7 Experimental Results

Evaluations of heuristic algorithms are usually based on extensive experimental testings. We will take this approach in comparing the forementioned heuristic algorithms with respect to the running time and profile-reduction performance. In this section, we will test them on a collection of sparse matrices that arise from practical applications, and some conclusions will be drawn on their general performance.

In the literature, there have been some comparative studies done on bandwidth and profile reduction algorithms. In the survey article [Cuthill72], Cuthill performs experiments on some model grid problems. Among the Cuthill-McKee (CM), reverse Cuthill-McKee (RCM), King and Levy algorithms, it is reported that "the CM and RCM have performed generally best for small bandwidths, the Levy ... for small wavefronts, and the Levy and RCM ... for small profiles." However, Cuthill does not include execution time of the algorithms in her paper.

In [Bolstad73], Bolstad, Leaf, Linderman and Kaper compare the RCM and King algorithms for the class of matrices arising in finite element applications. They note that, for the problems tested, the King scheme produces a smaller envelope than RCM, but at the expense of more computation time. It is also observed that the performances of both algorithms depend crucially on the starting node.

However, no specific suggestion has been given to overcome this difficulty.

Recently, Gibbs, Poole and Stockmeyer [Gibbs74a] have developed a new algorithm (GPS) which is especially tailored for bandwidth reduction. A novel feature in the GPS algorithm is the notion of pseudo-peripheral nodes, which we have discussed in section 5.5. The algorithm first finds a pseudo-peripheral node R and a node R' in the last level $L_{l(R)}(R)$ of $\mathcal{L}(R)$ where the width of $\mathcal{L}(R')$ is smallest among those defined by nodes in $L_{l(R)}(R)$. It then combines the rooted level structures $\mathcal{L}(R)$ and $\mathcal{L}(R')$ to form a new structure whose width is usually smaller than that of $\mathcal{L}(R)$ and $\mathcal{L}(R')$. The nodes in the graph are then numbered level by level in accordance with this new level structure. For a complete description of the algorithm, see [Gibbs74a].

In another paper [Gibbs74b], Gibbs, Poole and Stockmeyer provide by far the most extensive and complete comparative work on reduction algorithms. They compare the popular RCM and King algorithms with their new algorithm and those of [Cheng73a], [Collins73] and [Wang73]. The execution time of the GPS algorithm is impressive, and yet GPS typically produces bandwidths and profiles comparable to that of the RCM and King schemes. However, in their testing, the original version of RCM [Everstine72] is used, where a time-consuming search for a reasonably good starting node is performed. A similar search is involved in their implementation of the King algorithm.

In this connection, we pose the following question. What would be the performance of the RCM and King algorithms with pseudo-peripheral starting nodes? We tabulate results using such combinations, and it is interesting to compare the tables presented here with those given in [Gibbs74b]. In view of the excellent performance of the GPS algorithm, we also include it in our experiments.

The set of test problems is the same as those in [Gibbs74b]. It is a collection of sparse matrices by E.H. Cuthill and G.C. Everstine of the Naval Ship Research and Developement Centre. These matrices arise in actual application problems --- in the study of various structures, including aircraft, gas tanks, submarines, propellar blades and satellites. The paper [Gibbs74a] contains a detailed description of this collection.

The production code in [Crane75] is used for the GPS algorithm. It is the same program from which the data in [Gibbs74b] are obtained. All the other algorithms are programmed by the author in FORTRAN. A listing of these programs can be found in the appendix of this thesis. Extra care has been taken in the coding of the algorithms so that the production codes are quite uniform in style and performance. It is believed that the running time of the codes truly reflects the complexity of the algorithms. All tests are run on the University of Waterloo IBM 360/75 computer using the FORTRAN IV H compiler with OPT = 2. The

sizes of the production codes for the various algorithms are listed for comparison:

GPS	King	Levy	RCM	
(8.94	6.67	1.51	5.56	) K bytes.

A major portion of the codes for GPS, King and RCM is for finding pseudo-peripheral nodes.

The experimental results are tabulated in a set of tables. Table 5.1 contains the ordering time of the various algorithms. Note that the overall time for the GPS, CM, RCM and King algorithms is the sum of the time for finding a starting node and the actual ordering time of the respective schemes. Table 5.2 tabulates the resulting profiles and the corresponding factorization operation counts of the schemes. For completeness, we include the bandwidth and wavefront of the orderings in table 5.3.

			ORDERING TIME (seconds)					
			GPS		KING / RCM / CM			LEVY
|X|	m	|E|	Start node	Order	Start node	King Order	(R)CM Order	Order
68	8	134	0.038	0.034	0.029	0.038	0.022	0.037
90	8	299	0.061	0.043	0.054	0.043	0.024	0.058
92	8	272	0.043	0.045	0.045	0.046	0.027	0.059
130	9	509	0.070	0.056	0.055	0.068	0.032	0.101
159	9	501	0.076	0.087	0.071	0.072	0.039	0.126
174	10	464	0.102	0.076	0.094	0.076	0.038	0.158
185	14	740	0.119	0.078	0.100	0.103	0.043	0.178
220	8	784	0.093	0.116	0.073	0.087	0.058	0.217
263	13	911	0.174	0.128	0.162	0.113	0.054	0.329
263	8	698	0.131	0.130	0.108	0.109	0.054	0.318
310	10	1069	0.158	0.150	0.133	0.131	0.061	0.416
312	14	1119	0.222	0.156	0.149	0.163	0.075	0.441
346	18	1440	0.232	0.222	0.173	0.198	0.092	0.521
360	11	1321	0.123	0.177	0.125	0.228	0.072	0.591
436	11	1568	0.463	0.243	0.440	0.233	0.095	0.758
512	14	1495	1.007	1.018	0.947	0.362	0.260	1.018
555	14	2032	0.227	0.268	0.198	0.458	0.116	1.495
918	12	3233	0.798	0.533	0.752	0.841	0.178	3.342

Table 5.1 Ordering time for algorithms.

N	PROFILE					FACT OPERATIONS				
	GPS	King	Levy	RCM	CM	GPS	King	Levy	RCM	CM
68	269	216	216	269	271	1114	735	735	1114	11
90	579	575	569	575	608	2808	2779	2738	2779	30
92	736	673	756	739	839	4315	3670	4628	4384	55
130	1588	1496	2373	1562	1601	13286	11912	28994	12874	135
159	971	1261	1215	965	1041	4837	7933	7575	4816	55
174	1466	1338	1342	1462	1647	8835	7445	7487	8782	108
185	3610	3993	3982	3711	4096	44045	54593	55799	46811	558
220	1868	1754	1749	1809	1890	11182	10228	10200	10672	113
263	2346	2158	2449	2407	2743	14901	12687	16855	15882	208
263	2001	1827	1861	2132	2231	10953	9378	9663	12573	136
310	2726	2694	3712	2725	2801	16598	16200	32005	16579	174
312	5548	4306	4036	5548	7174	63934	41184	36068	63934	1044
346	7650	6457	12612	7688	10079	105649	75011	304909	106929	1810
360	6364	9884	10362	6364	6937	78068	179107	195053	78068	922
436	7844	7928	7510	8181	9359	94687	97389	86959	101787	1321
512	4669	4786	5315	4749	5981	41797	44398	51566	42681	636
555	28976	28382	32494	28807	37565	918328	894509	1195888	924441	1547
918	20369	54175	49204	21113	24259	281037	1988911	1749462	304544	3979

Table 5.2 Profile and operation count of schemes.

	BANDWIDTH				WAVEFRONT				
N	GPS	King	Levy	(R)CM	GPS	King	Levy	RCM	CM
68	7	30	48	7	7	5	5	7	7
90	7	11	17	9	7	8	7	8	9
92	13	38	28	14	12	11	13	12	14
130	18	25	60	19	18	17	29	17	19
159	12	65	66	12	11	17	17	11	12
174	13	27	29	13	11	9	9	12	13
185	29	86	150	31	28	31	34	31	31
220	12	13	63	13	12	10	10	10	13
263	19	29	71	21	16	14	20	18	21
263	14	23	35	14	14	9	9	14	14
310	14	23	101	14	13	13	20	13	14
312	37	142	94	37	28	25	23	28	37
346	46	67	341	47	34	26	62	34	47
360	34	303	292	34	34	46	48	34	34
436	33	49	69	34	33	33	33	34	34
512	29	66	231	30	25	26	31	27	30
555	91	463	399	107	81	86	96	85	107
918	49	736	669	50	39	104	95	43	50

Table 5.3 Bandwidth and Wavefront of schemes.

5.8 Comparative Remarks

From the set of tables in section 5.7, we offer the following general observations and remarks on the ordering algorithms.

A substantial portion of the overall ordering time for the GPS, King and CM/RCM algorithms is spent in finding a pseudo-peripheral node. On the average, GPS, King and CM/RCM use about 53%, 50% and 68% respectively of the total time for this purpose. Can a reasonably good starting node be determined using significantly less time? This question deserves further investigation.

Although the Levy algorithm does not require any particular node to start the ordering, it typically requires more execution time than the others for problems of substantial size. For example, it is four times slower than the RCM scheme for the problem with N = 555. As we have noted in section 5.3.1, its main shortcoming is its slowness.

The CM/RCM algorithm is the fastest in all cases. On comparing with the data in [Gibbs74b], this indicates a tremendous increase in speed when the algorithm is to be started at some pseudo-peripheral node. Similar speed-up can be noticed in the King algorithm.

Such an increase in speed for the RCM and King algorithms would not be useful if their profile-reducing

abilities were at the same time decreased. Fortunately, the results in tables 5.2 and 5.3 show that the generated bandwidths and profiles are comparable to that from the original versions of the algorithms [Gibbs74b].

To evaluate the algorithms with respect to their profile-reducing ability, we choose to measure their deviations from the minimum profile. Since the minimum profile is not known, the profile minimum among the five algorithms is taken as such. For each algorithm, we compute its arithmetic mean (average) of the % difference from the lowest profile, and its standard deviation. The standard deviation values serve to measure the consistency of their performances.

	PROFILE					OPCOUNT				
	GPS	KING	LEVY	RCM	CM	GPS	KING	LEVY	RCM	CM
average % difference	7.4	15.2	26.7	8.3	22.8	15.0	47.5	78.7	17.6	50.2
standard deviation	9.7	39.0	38.2	9.8	18.1	20.5	139.5	132.2	20.7	45.7

Table 5.4: Average performance of algorithms.

For our set of test matrices, it is found that on the average, the GPS and RCM algorithms deviate least from the smallest profile, and perform most consistently. Although the King algorithm produces the lowest profiles in a lot of cases, its erratic behavior in a few examples, notably when N is 360 and 918, brings it down in the evaluation.

The Levy algorithm also exhibits erratic results for some test problems. It is interesting to note that it performs poorly on the graph with N = 346, which is disconnected.

The comparative analysis in section 5.4.2 shows that RCM can never be worse than CM in the envelope context. From table 5.2, we note that RCM actually reduces the envelope sizes and envelope operation counts in all test examples. The difference can be considered as significant; indeed, the reduction can be as much as 25% for profile and 40% for arithmetic operations. More dramatic savings of RCM over CM on model problems have been reported by Liu and Sherman [Liu75].

To summarize our observations, the RCM algorithm shows consistently good performance in profile reduction. It produces profiles which are, at worst, only slightly larger than the "minimum" profile. Together with its simplicity and speed, it is a practical choice for a profile ordering.

The GPS algorithm exhibits similar consistency as a profile minimizer in all the test matrices. However, for

some finite element problems which we will report in chapter 6, its performance is less effective. On the other hand, it does a good job in finding an ordering with a small bandwidth. Indeed, it is an excellent choice for finding a good bandwidth ordering.

Chapter 6 On Reducing the Profile of Finite Element Systems

In this chapter, we shall consider the profile reduction problem for the class of finite element graphs. Such graphs have special structures which can be exploited for our purpose. Along this line, we introduce the idea of an annihilation sequence, which originates with the work of Irons. A profile reduction algorithm for finite element graphs is proposed and its performance is compared with that of the general graph algorithms in chapter 5 via extensive experimental testings.

6.1 Finite Element Systems of Equations

The **finite element method** is a powerful numerical method for solving partial differential equations. The method has many important application areas, such as structural mechanics, wave propagation, elastic stability, heat conduction, and fluid mechanics. The fundamental basis in the applicability of the method is the existence of a variational principle in the formulation of the problem.

In this chapter, we shall study the solution of sparse symmetric positive definite linear systems that arise in the application of finite element methods to two dimensional problems. We begin with the few aspects of the method that concern us. [George71], [Strang73], and [Zienkiewicz70] are good sources concerning the method.

Let R be a planar region with boundary ∂R. In solving a two-dimensional boundary value problem by a finite element method, the region R is covered with a mesh M of triangular elements. The mesh M has a node at each vertex and may also have nodes lying on each edge and/or in the interior of each element. Following [George71], we use the notation M(r,s,t)† to refer to the mesh M, where r, s, and t are respectively the numbers of unknowns associated with a vertex, an edge and an interior. We assume that $r \geq 1$. The system of triangles M(r,s,t) is called a finite element mesh and M is the underlying mesh.

A finite element system or mesh system of equations associated with M(r,s,t) is any N by N symmetric positive definite linear system

$$A x = b, \qquad (6.1)$$

where the entry A_{ij} is nonzero if and only if unknowns x_i and x_j are associated with nodes of the same mesh element of M. Here, N is the total number of unknowns in the finite element mesh M(r,s,t). Figure 6.1 is an example of a finite element mesh M(1,1,0) and its associated 16 by 16 mesh matrix.

---

† The numbers r, s, and t depend on the degree of the polynomial and the type of interpolation used in the finite element approximation.

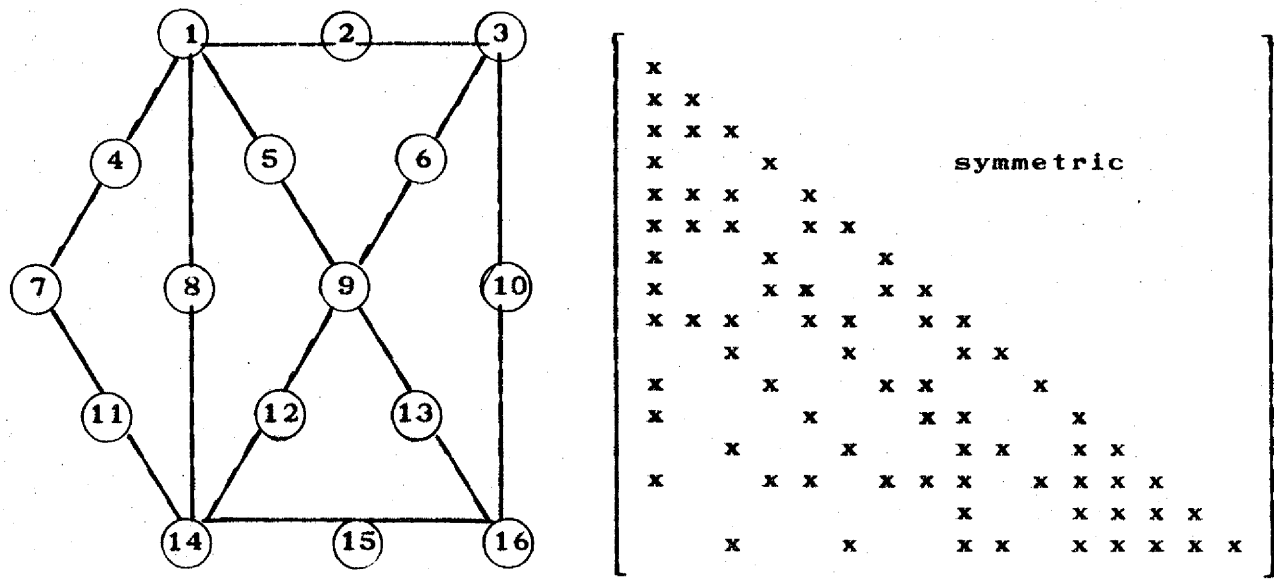


Figure 6.1 A finite element mesh and its associated matrix.

The graph G(A) of a finite element matrix bears a close relation with the finite element mesh, and it can be reconstructed from the mesh easily. From definition (6.1), the nodes of each triangular element in M(r,s,t) forms a complete subgraph or a clique. To obtain G(A), we merely add edges so that mesh nodes belonging to the same element are joined. We shall refer to G(A) as a finite element graph. The complete subgraph for an element in figure 6.1 is shown in figure 6.2.

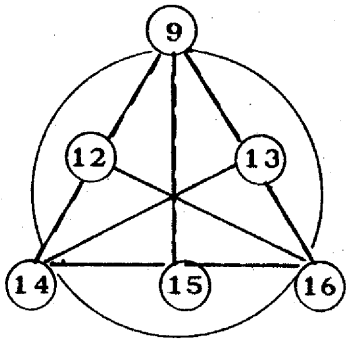


Figure 6.2 Graph for a basic element.

In general, the graph of the coefficient matrix has many more edges than the diagram of the finite element mesh, and very often, the graph is non-planar. Since the finite element mesh is already adequate in showing the zero-nonzero structure of the matrix, we shall use it to represent the matrix diagramatically. Birkhoff and George [Birkhoff73] call it a clique diagram. In [George73a], George establishes an interesting correspondence between the elimination process and a sequence of clique diagrams.

.2 Element Annihilation and its Relation to Ordering

Given a finite element mesh M(r,s,t), let G be the finite element graph associated with it. We are interested in the following questions. How do we choose an ordering α so that the ordered graph G_α has a minimum or near minimum profile? How can the structure of the underlying mesh M be exploited in order to determine such an ordering α?

A practical answer may be the use of element annihilation which was first considered by Irons. In [Irons70], he has exploited this idea in his frontal approach to finite element systems. The technique alternates between assembly of element coefficients and elimination. This reduces the size of the matrix to be kept in core and leads to impressive results. Irons also points out that the node numbering is not critical but the way in which the elements are ordered is. In this section, we shall make a systematic study of the effect of element annihilation in profile and frontal schemes.

We begin with some definitions. The dual graph of a mesh M is the undirected graph $(\mathcal{T}(M),\mathcal{E}(M))$, where $\mathcal{T}(M)$ consists of all the triangular elements of M and for elements T and T' of M, $\{T,T'\}$ is in $\mathcal{E}(M)$ if and only if T and T' share a common side. Consider the region with mesh M in figure 6.3. Its corresponding dual graph is given as shown.

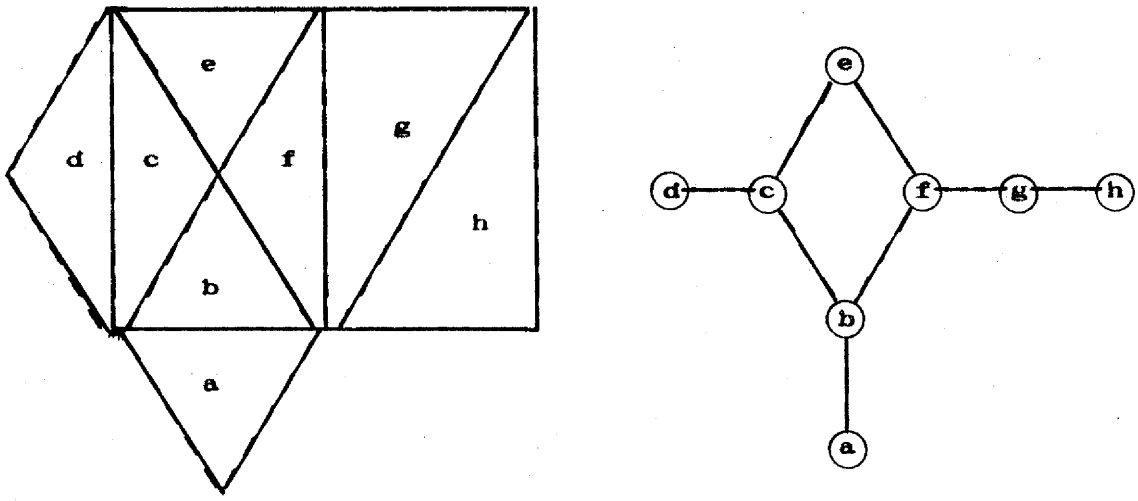


Figure 6.3 Mesh and its dual graph

The following is a trivial and yet important property of dual graphs.

Lemma 6.2 For any T in $\mathcal{T}(M)$, the degree of T in the dual graph is less than or equal to 3. □

An ordering on the triangular elements of the dual graph of the mesh M is defined to be an annihilation sequence. In what follows, we shall show that any annihilation sequence σ defines a class of orderings on the corresponding finite element graph associated with M(r,s,t).

Let $T_1, T_2, \ldots, T_m$ be the sequence of triangular elements in M ordered as given by σ. Here, $m = |\Gamma(M)|$. For our purpose, we let each T_i to be the set of nodes associated with the corresponding triangle. The *i-th annihilated set* can then be defined as

$$an(\sigma, T_i) = T_i \setminus \bigcup_{k=i+1}^{m} T_k.$$

Node orderings are induced as follows.

```
procedure INDUCE(σ, α);
begin
   for i:=1 step 1 until m do
   begin
      for x ∈ an(σ,T_i) do
         number x in α
   end
end;
```

When the annihilation sequence is clear from the context, we use $an(T_i)$ to refer to $an(\sigma, T_i)$. However, it should be noted that the annihilated set of variables depends on the given sequence σ.

Observation 6.3 The set $an(\sigma, T_i)$ is precisely the set of nodes removed when T_i is annihilated from the remaining triangles.

Lemma 6.4: $x \in an(\sigma, T_i)$ if and only if

$$Adj(x) \subset \bigcup_{k=1}^{i} T_k, \text{ and } x \in T_i. \qquad \square$$

After step i, the set of numbered nodes is given by the set

$$\bigcup_{k=1}^{i} an(T_k).$$

We find it convenient in subsequent discussions to denote this set by $an(\sigma, T_1, \ldots, T_i)$ or $an(T_1, \ldots, T_i)$ if σ is clear from context. Evidently, an annihilation sequence σ induces a class of node orderings which can be represented by the series of node subsets:

$$an(T_1),\ an(T_2),\ \ldots,\ an(T_m).$$

In the remaining part of this section, we shall see how the nodes in $an(T_i)$ should be numbered internally.

At step i of the annihilation process, let $\{T_1, \ldots, T_{i-1}\}$ be the set of triangles that have been annihilated. An element T in $\{T_1, \ldots, T_{i-1}\}$ is said to be pseudo annihilated if $T \cap an(T_1, \ldots, T_{i-1}) = \emptyset$. In other words, a pseudo annihilated element is one that has been annihilated and yet none of its nodes has been numbered (or removed). This definition is motivated from the following result.

Theorem 6.1: Let $\sigma: T_1, T_2, \ldots, T_m$ be an annihilation sequence on a finite element mesh M(r,s,t). At step i, if no pseudo annihilated element has a common node with $an(T_i)$

Lemma 6.4: $x \in an(\sigma, T_i)$ if and only if

$$Adj(x) \subset \bigcup_{k=1}^{i} T_k, \text{ and } x \in T_i. \qquad \square$$

After step i, the set of numbered nodes is given by the set

$$\bigcup_{k=1}^{i} an(T_k).$$

We find it convenient in subsequent discussions to denote this set by $an(\sigma, T_1, \ldots, T_i)$ or $an(T_1, \ldots, T_i)$ if σ is clear from context. Evidently, an annihilation sequence σ induces a class of node orderings which can be represented by the series of node subsets:

$$an(T_1),\ an(T_2),\ \ldots,\ an(T_m).$$

In the remaining part of this section, we shall see how the nodes in $an(T_i)$ should be numbered internally.

At step i of the annihilation process, let $\{T_1, \ldots, T_{i-1}\}$ be the set of triangles that have been annihilated. An element T in $\{T_1, \ldots, T_{i-1}\}$ is said to be <u>pseudo annihilated</u> if $T \cap an(T_1, \ldots, T_{i-1}) = \emptyset$. In other words, a pseudo annihilated element is one that has been annihilated and yet none of its nodes has been numbered (or removed). This definition is motivated from the following result.

Theorem 6.1: Let $\sigma: T_1, T_2, \ldots, T_m$ be an annihilation sequence on a finite element mesh M(r,s,t). At step i, if no pseudo annihilated element has a common node with $an(T_i)$

then the profile is independent of the ordering of the nodes in $an(T_i)$.

Proof: By theorem 3.2, it is sufficient to show that for any x and y in $an(T_i)$,
$Adj(an(T_1,..,T_{i-1})\cup\{x\})\cup\{x\} \subset Adj(an(T_1,..,T_{i-1})\cup\{y\})\cup\{y\}$.
Clearly, by lemma 6.3, x and $y \notin an(T_1,..,T_{i-1})$. Consider any z in the right hand side.

If $z = x$, then $x \in Adj(y) \subset Adj(an(T_1,..,T_{i-1})\cup\{y\})$, since x and y belong to the element T_i.

On the other hand, if $z \in Adj(an(T_1,..,T_{i-1})\cup\{x\})$, we need only to consider the case $z \in Adj(x)$. By lemma 6.4, x and $z \in T_k$ for some $1\le k\le i$. If $z \in T_i$, evidently $z \in Adj(y)\cup\{y\}$. But if $z \in T_k$ where $k<i$, then $x \in T_k \cap an(T_i)$; this means at this stage T_k is not a pseudo annihilated element. Thus $T_k \cap an(T_1,..,T_{i-1}) \neq \emptyset$ and

$$z \in Adj(T_k \cap an(T_1,..,T_{i-1})) \subset Adj(an(T_1,..,T_{i-1})).$$

The result follows. □

Theorem 6.1 provides a condition, whereby all the node orderings induced by σ are equivalent, in the sense that they give the same envelope size. The following corollary contains a simpler and more useful condition for the result of theorem 6.1 to hold.

Corollary 6.5: Let M(r,s,t) and σ be the same as in theorem 6.1. If $an(T_i)$ is non-empty for $1\le i\le m$, then all the orderings induced by σ are equivalent.

Proof: Note that $an(T_i) \neq \emptyset$ implies the element T_i has never been pseudo annihilated. □

The condition in corollary 6.5 is satisfied in many situations; for example, when the mesh M(r,s,t) has interior nodes (i.e. t is nonzero). But the conditions in theorem 6.1 or its corollary are not always satisfied. Consider the mesh M and annihilation sequence in figure 6.4. M(1,0,0) is assumed. It is easy to see that $an(T_1)$ is empty and the two different orderings in figure 6.4 on nodes of $an(T_2)$ result in different envelope sizes.

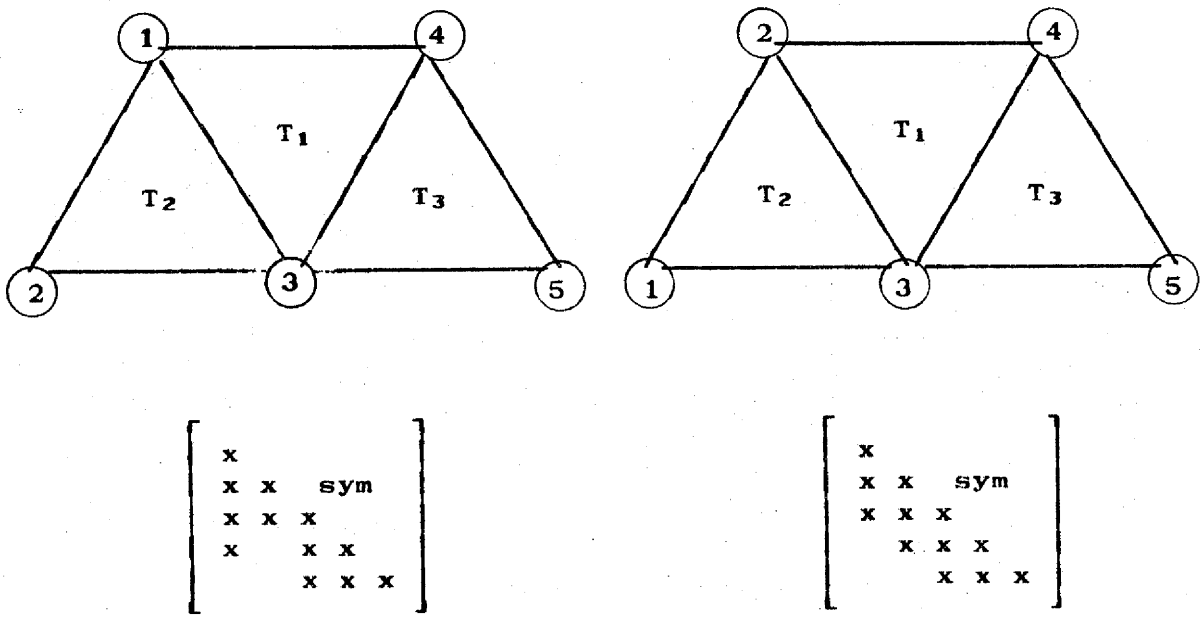


Figure 6.4: Two different induced orderings

Before going to the next section, we illustrate the idea of <u>induced ordering</u> using the example in figure 6.3. Consider the following annihilation sequence of the dual graph:

a, b, c, d, e, f, g, h.

We assume that quadratic interpolation is used so that the corresponding finite element mesh is M(1,1,0). Note that $an(T_i)$ is non-empty for $1 \leq i \leq 8$. Corollary 6.5 thus applies and we can number nodes in $an(T_i)$ in any order.

The complete annihilation process is depicted in figure 6.5. The overall induced ordering is given in figure 6.6. The envelope size of the corresponding permuted finite element system turns out to be 116.

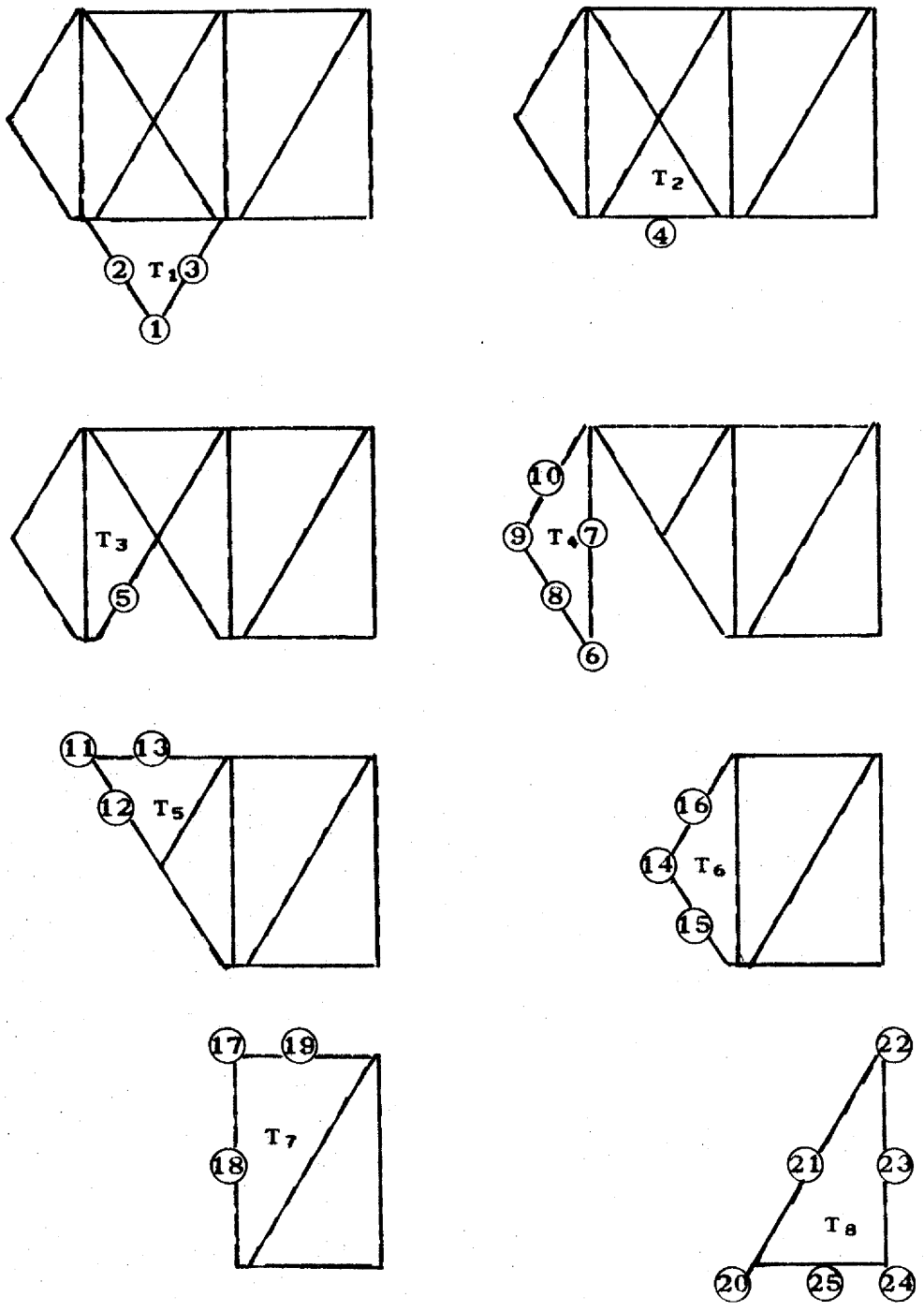
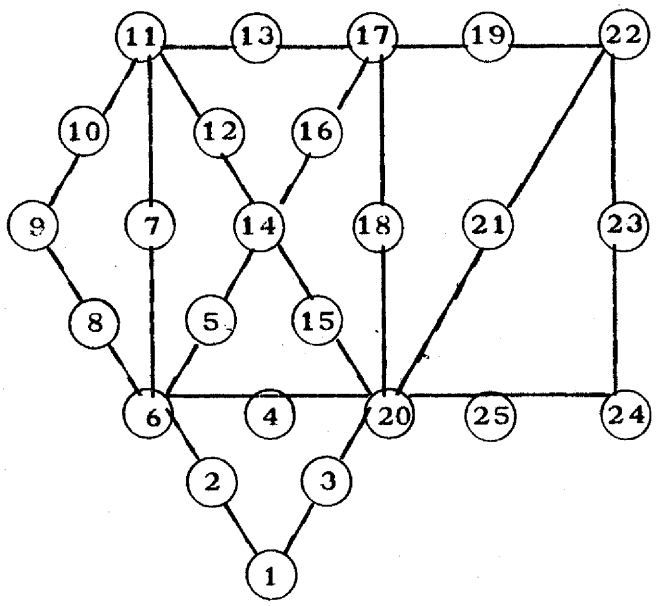


Figure 6.5: Annihilation process.



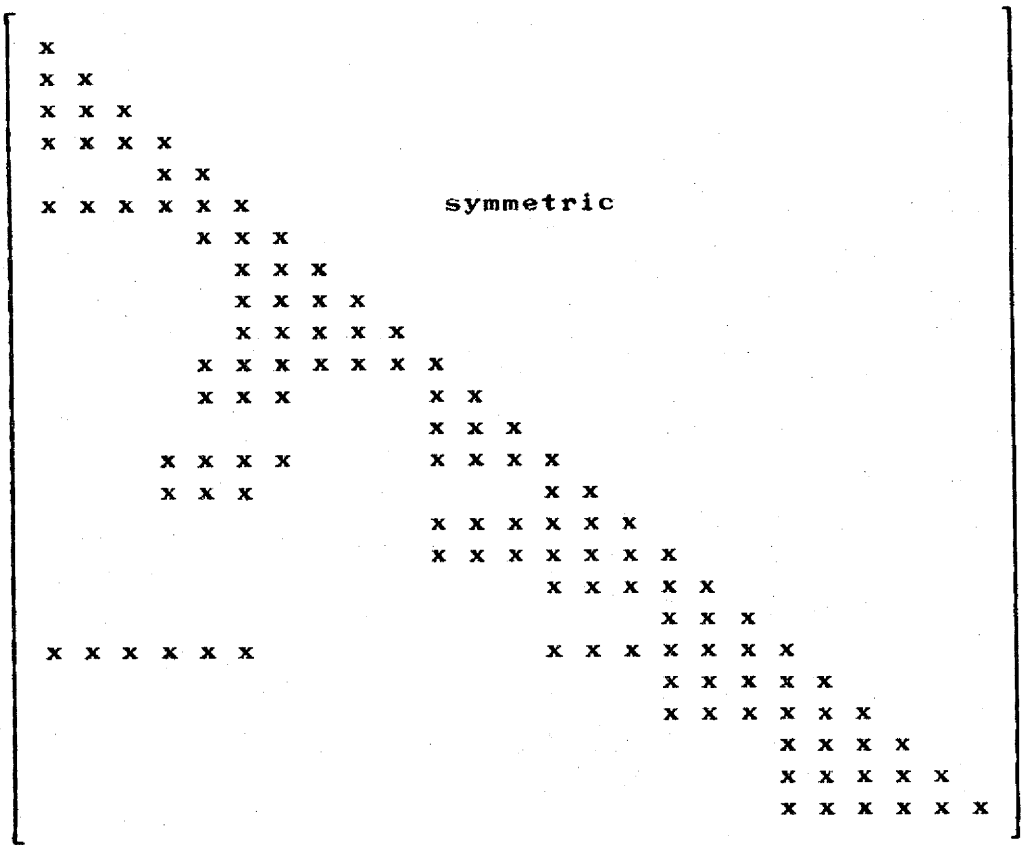


Figure 6.6: Overall ordering and the ordered matrix

6.3 Existence and Characterization of a Profile-Minimizing Annihilation Sequence

Our primary aim is to minimize the envelope size or the profile of the graph associated with any given finite element mesh. The following results justify the use of annihilation sequences in our analysis.

Theorem 6.2: If α is any ordering on the finite element graph G, there exists an annihilation sequence σ which induces an ordering $\bar{\alpha}$ satisfying

$$|Env(G_{\bar{\alpha}})| \leq |Env(G_{\alpha})|,$$

and $$\theta_E(G_{\bar{\alpha}}) \leq \theta_E(G_{\alpha}).$$

Proof: We shall prove the theorem by constructing an annihilation sequence σ such that one of its induced ordering $\bar{\alpha}$ is never inferior to α in terms of storage and operation count.

Let y_1 be the node numbered first in α. This node y_1 belongs to some triangular element, say T. Assign T to be the first element in our sequence σ. Then identify the set an(T). We note that for any x in an(T),

$$Adj(x) \cup \{x\} \subseteq Adj(y_1) \cup \{y_1\}.$$

Define a new ordering α_1, which renumbers the nodes in an(T) first (in the same order as they appear in α), while keeping the remaining nodes in the same relative order as in α. The set an(T) may be empty. By theorem 3.2,

$$|Env(G_1)| \leq |Env(G_{\alpha})|,$$

and $$\theta_E(G_1) \leq \theta_E(G_{\alpha}),$$

where G_1 is the graph ordered by α_1.

We can now consider the ordering on the remaining mesh with T_1 (i.e. T) removed; the new ordering α_1 will be used instead. We follow the same argument as before to determine element T_2 in σ. In this case, y_2, the node immediately after $an(T_1)$ in α_1, will be considered. Again, we have for $x \in an(T_2)$,

$$Adj(an(T_1)\cup\{x\}) \cup \{x\} \subseteq Adj(an(T_1)\cup\{y_2\}) \cup \{y_2\}.$$

Let α_2 be the ordering thus obtained.

We can continue the process until σ is completely constructed. If $m = |\bar{T}(M)|$, the induced ordering α_m of σ satisfies

$$|Env(G_m)| \leq |Env(G_\alpha)|,$$

and $$\theta_E(G_m) \leq \theta_E(G_\alpha).$$

This completes the proof of the theorem. □

Theorem 6.3: Let M(r,s,t) be a finite element mesh. Then there exists an annihilation sequence σ which induces a minimum envelope ordering.

Proof: Consider any minimum envelope ordering α on the system associated with M(r,s,t). Theorem 6.2 implies the existence of an annihilation sequence σ which induces an ordering $\bar{\alpha}$ such that

$$|Env(G_{\bar{\alpha}})| \leq |Env(G_\alpha)|.$$

Thus $\bar{\alpha}$ itself is necessarily a minimum envelope ordering. □

Theorem 6.4: If t is nonzero, any minimum envelope ordering α for a mesh system associated with M(r,s,t) is an induced ordering of some annihilation sequence.

Proof: Assume that α: $x_1, x_2, \ldots, x_N$ is a minimum envelope ordering and that it is different from all induced orderings. Apply the method in the proof of theorem 6.2 to α, and let $\bar{\alpha}$: $y_1, y_2, \ldots, y_N$ be the induced ordering obtained. By construction and the minimality of α, we have

$$|Env(G_{\bar{\alpha}})| = |Env(G_{\alpha})|,$$

Since α is different from all induced ordering, let i be the index such that $x_k = y_k$ for $1 \le k < i$, and $x_i \neq y_i$. The nodes x_i and y_i can only be different if

$$\{x_1, \ldots, x_{i-1}\} = an(T_1, \ldots, T_j)$$

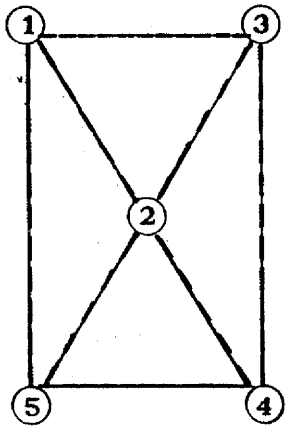
for some j, and $y_i \in an(T_{j+1})$ but $x_i \notin an(T_{j+1})$. This means the removal of x_i at this step annihilates T_{j+1} and some other triangular elements. Since $t \neq 0$, we have

$$Adj(\{x_1, \ldots, x_{i-1}, y_i\}) \cup \{y_i\} \subsetneq Adj(\{x_1, \ldots, x_{i-1}, x_i\}) \cup \{x_i\}.$$

By corollary 3.7, α cannot be a minimum envelope ordering. □

We conjecture that theorem 6.4 holds if s+t is nonzero. But when s+t is equal to zero, the example M(1,0,0) in figure 6.7 shows a minimum ordering which does not correspond to any induced ordering.

Theorem 6.3 assures the existence of an annihilation sequence that will yield a minimum envelope node ordering



$$\begin{bmatrix} x & & & & \\ x & x & & \text{sym} & \\ x & x & x & & \\ & x & x & x & \\ x & x & & x & x \end{bmatrix}$$

Figure 6.7: Minimum ordering on M(1,0,0)

on the given finite element mesh. Furthermore, in many important cases, we know from theorem 6.4 (and the conjecture after the theorem) that finding annihilation sequences is in fact a possible approach. These theoretical observations motivate the study of element orderings on the dual graph of M. We begin the study by characterizing envelope-minimizing annihilation sequences from the structure of the dual graph.

Recall that a node ordering $\alpha: x_1, x_2, \ldots, x_N$ is minimum if it minimizes the quantity $|Env(G_\alpha)|$ over all possible orderings. What, then, is the criterion we should use in seeking annihilation sequences?

Let us consider the stage when some triangles in M have already been annihilated. A node is "active" (i.e. in the front) if and only if it belongs to some annihilated

elements as well as some remaining triangles. More precisely, if the annihilation of triangles $T_1,\ldots,T_k$ removes nodes $x_1,\ldots,x_l$, then

$$x \in Adj(\{x_1,\ldots,x_l\}) \text{ if and only if } x \in \left(\bigcup_{j=1}^{k} T_j\right) \cap \left(\bigcup_{j=k+1}^{m} T_j\right).$$

Thus, the number of active nodes, $|Adj(\{x_1,\ldots,x_l\})|$, is usually governed by the present number of active triangle sides. By an active side, we mean an edge in the set $Inc(\{T_1,\ldots,T_k\})$ of the dual graph. (See definition 3.2 in section 3.1). Although there are other factors governing the number of active nodes, we find that minimizing the quantity $|Inc(\{T_1,\ldots,T_k\})|$ normally reduces the size of node front. Thus, we arrive at the following observation: a sequence T_1, T_2, ..., T_m that minimizes the sum of edge fronts

$$\sum_{k=1}^{m} |Inc(\{T_1, \ldots, T_k\})| \qquad (6.2)$$

induces near-minimum envelope node orderings. The set $Inc(\{T_1,\ldots,T_k\})$ will be called the k-th <u>edge front</u>, while the quantity $|Inc(\{T_1,\ldots,T_k\})|$ the k-th <u>edge frontwidth</u>.

While the criterion function (6.2) is associated with the dual graph which is structurally simpler and is often much smaller in size, we note that sequences minimizing the sum of edge fronts do not necessarily induce minimum envelope node orderings. Consider the finite element mesh M(1,0,0) in figure 6.8. It is not difficult to see that the minimum value of

$$\Sigma \, |\mathrm{Inc}(\{T_1, \ldots, T_k\})|$$

over all sequences is 8. The two annihilation sequences σ_1 and σ_2 both attain this minimum.

However, we find that

$$|\mathrm{Env}(G_1)| = 17,$$

and $$|\mathrm{Env}(G_2)| = 18,$$

where G_1 and G_2 are the graphs labelled by σ_1 and σ_2 respectively. Thus, the sum of edge fronts for σ_2 is minimum and yet its induced node ordering does not give a minimum envelope.

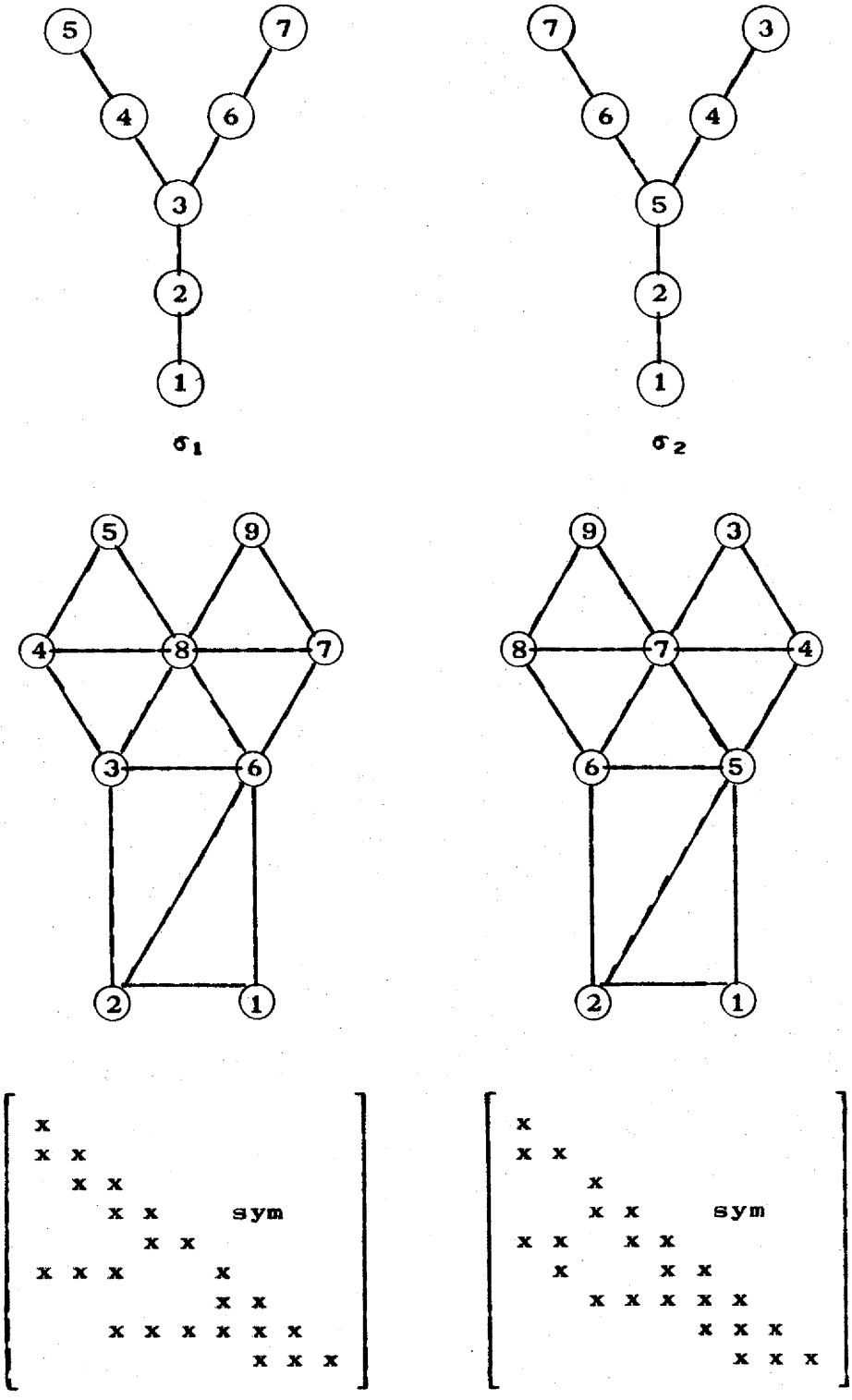


Figure 6.8: Sequences that are not equivalent.

6.4 Ordering Algorithms

6.4.1 Improved Ordering by Element Annihilation

The proof of theorem 6.2 in section 6.3 provides a method that improves any pre-determined node ordering of a finite element graph. Essentially, any given ordering determines in a natural way an annihilation sequence, which in turn induces a node ordering. We can describe the algorithm formally as follows.

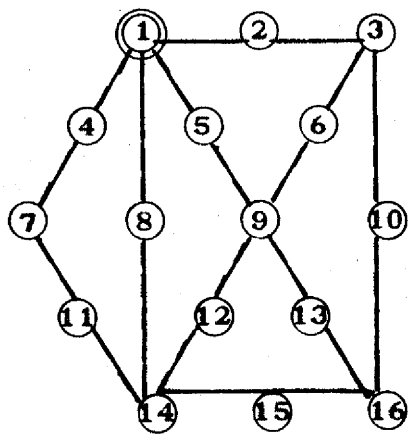
```
procedure IMPROVE(X, T, α, ᾱ);
begin
   comment: The input ordering α is improved by
                element annihilation to a new ordering ᾱ;
   i:= 0;
   while i < |X| do
   begin
      x:= first un-renumbered node in α;
      for triangular element T ⊃ {x} do
      begin
         for y ∈ an(T) do
         begin
            i:= i + 1;
            ᾱ(i):= y
         end
      end
   end
end.
```

With an appropriate data structure of the finite element graph, IMPROVE is extremely cheap, $O(|X|,|\bar{\Upsilon}|)$. We will consider the computer representation of finite element graphs in section 6.5.1.

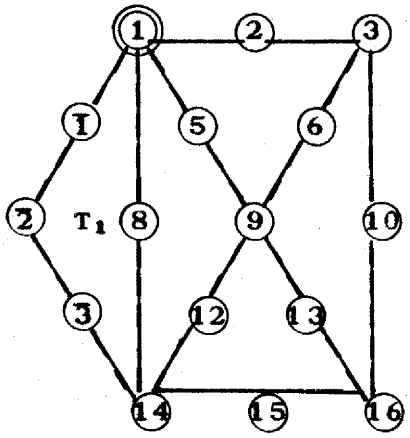
From theorem 6.2, the induced ordering $\bar{\alpha}$ can never increase the profile nor the envelope operation count. Consider the application of IMPROVE to the initial ordering in figure 6.1. Figure 6.9 illustrates the way $\bar{\alpha}$ is formed. In the figure, we use $\bar{I}$ to denote the i-th node numbered in $\bar{\alpha}$. This example demonstrates actual storage and operation reductions over the initial ordering.

The best choice of the initial ordering seems to be the one obtained from the reverse Cuthill-McKee (RCM) algorithm, which is known to be fast and effective (see chapter 5). The correspondingly induced ordering shall be called the improved RCM (IRCM) ordering. Numerical experiments have been performed on the n by n regular right triangular mesh M(1,1,0), and the results are tabulated in table 6.1. They show that savings, though not impressive, can actually be achieved over the RCM ordering.

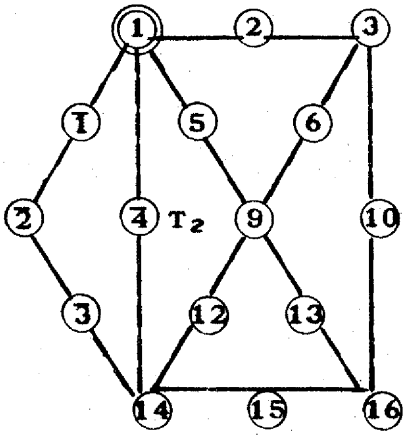
Similar experiments have been tried to improve the King algorithm. It is interesting to see that, in all test runs, no improvement can be found. It may be attributed to



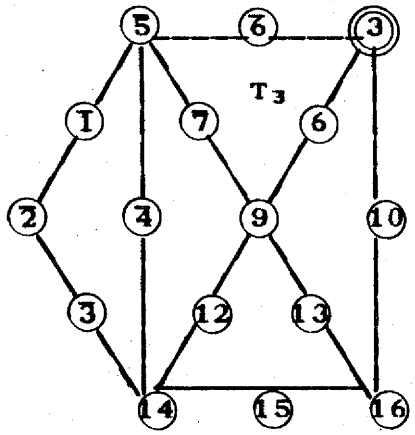
Original α



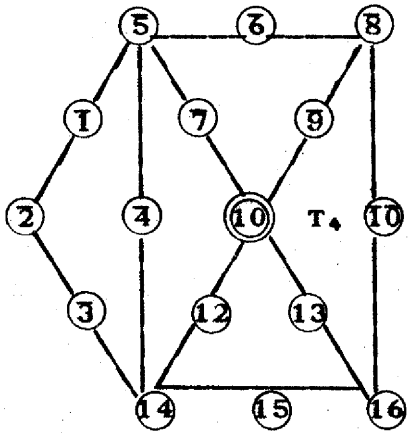
α_1



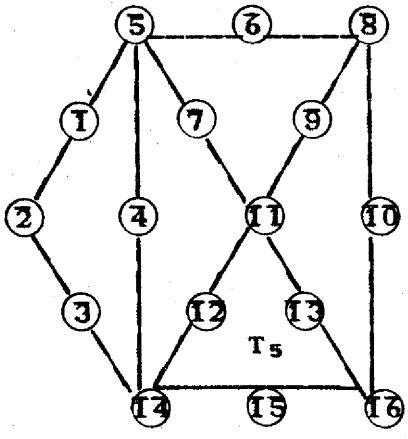
α_2



α_3



α_4



$\bar{\alpha}=\alpha_5$

Figure 6.9: Improved ordering by annihilation.

PROBLEM			PROFILE			OPERATION COUNT		
n	N	Nonz	RCM	IRCM	IRCM/RCM	RCM	IRCM	IRCM/RCM
2	25	96	128	116	0.9063	589	484	0.8217
3	49	207	331	309	0.9335	1782	1570	0.8810
4	81	360	674	640	0.9496	4183	3810	0.9110
5	121	555	1189	1141	0.9596	8324	7740	0.9298
6	169	792	1908	1844	0.9664	14857	13992	0.9418

Table 6.1: RCM and IRCM on n by n regular right triangular mesh M(1,1,0).

the local-frontwidth-minimizing characteristic of the King algorithm, so that the corresponding node ordering usually turns out to be an induced ordering of some annihilation sequence. This may help to account for the better performance of the King algorithm over RCM in finite element applications as reported in [Bolstad73].

6.4.2 Annihilation Sequence Minimizing Edge-Frontwidth

In view of the discussion at the end of section 6.3, we address ourselves to the problem of finding annihilation sequences for the dual graph such that

$$\sum |\mathrm{Inc}(\{T_1, \ldots, T_k\})|$$

is minimum or near minimum. It bears a great similarity to the envelope-minimizing problem, where the sum

$$\sum |\mathrm{Adj}(\{x_1, \ldots, x_i\})|$$

is minimized in a symmetric graph. This suggests an algorithm similar to that of King (section 5.3.2). We now describe the new scheme for connected dual graphs below. Extension to general dual graphs is straightforward.

```
procedure FIND_AS(T, E, T0, σ );
begin
   comment: (T,E) is the input dual graph, T0 the given
            starting element, and σ the output annihilation
            sequence;
   i:= 1
   σ(1):= T0;
   while i < |T| do
   begin
      T:= a triangle in Adj({σ(1),..,σ(i)});
      for T̂ ∈ Adj({σ(1),..,σ(i)}) do
      begin
         if |Inc({σ(1),..,σ(i),T̂})| < |Inc({σ(1),..,σ(i),T})|
            then T:= T̂
      end;
```

```
        i:= i + 1;
        σ(i):= T
    end
end.
```

Like the King algorithm, the performance of FIND_AS is sensitive to the starting element T_0. The success in using pseudo peripheral node to start the King algorithm suggests the use of a pseudo peripheral element in the dual graph $(\mathcal{T},\mathcal{E})$ as T_0.

In this way, a node ordering can be obtained by first generating an annihilation sequence σ with FIND_AS, and then applying the technique of annihilation to σ. In subsequent discussions, this combination will be referred to as the element annihilation (EA) algorithm.

In FIND_AS, an element is labelled next in σ, if it increases the edge frontwidth the least. An equivalent condition, which is more oriented to implementation, is established below.

Lemma 6.6: Let $T \in Adj(\{T_1,..,T_i\})$. The quantity $|Inc(\{T_1,..,T_i,T\})|$ is minimum if and only if $|Inc(T)| - 2\,|Inc(T) \cap Inc(\{T_1,..,T_i\})|$ is minimum.

Proof: The lemma follows from:

$|Inc(\{T_1,..,T_i,T\})|$

$$= |Inc(\{T_1,..,T_i\})| - |Inc(\{T_1,..,T_i\}) \cap Inc(T)|$$
$$+ [\,|Inc(T)| - |Inc(\{T_1,..,T_i\}) \cap Inc(T)|\,]$$
$$= |Inc(\{T_1,..,T_i\})| + |Inc(T)| - 2|Inc(T) \cap Inc(\{T_1,..,T_i\})|$$

□

Lemma 6.7: Assume that there is no isolated element in the dual graph. If $|Inc(\{T_1,..,T_i,T\})|$ is minimum for T in $Adj(\{T_1,..,T_i\})$, then it is minimum over all unlabelled elements.

Proof: Assume that $T \in Adj(\{T_1,..,T_i\})$. Then we have

$$Inc(T) \cap Inc(\{T_1,..,T_i\}) \neq \emptyset,$$

so that

$$|Inc(\{T_1,..,T_i,T\})|$$
$$= |Inc(\{T_1,..,T_i\})| + |Inc(T)| - 2|Inc(T) \cap Inc(\{T_1,..,T_i\})|$$
$$\leq |Inc(\{T_1,..,T_i\})| + |Inc(T)| - 2$$
$$\leq |Inc(\{T_1,..,T_i\})| + 1.$$

On the other hand, for $T' \notin Adj(\{T_1,..,T_i\})$,

$$|Inc(\{T_1,..,T_i,T'\})|$$
$$= |Inc(\{T_1,..,T_i\})| + |Inc(T')|$$
$$\geq |Inc(\{T_1,..,T_i\})| + 1.$$

□

From lemma 6.7, we note that the element selected in FIND_AS is actually one that increases the edge frontwidth least among all unlabelled elements. The King node ordering algorithm does not have the corresponding property [Levy71].

.5 Comparative Study

.5.1 Data Management for Finite Element Graphs

The simplicity of data structures and the storage economy of data information are important aspects in implementing ordering algorithms for finite element graphs. A neat storage scheme simplifies the actual coding of the algorithm and often reduces the amount of overhead. On the other hand, less storage requirement makes it possible to order larger problems in primary memory. In order to compare data management of the ordering algorithms in section 6.4 and chapter 5, we have to look for relevant information of a finite element graph and their computer representations.

Triangular elements and nodes are the two basic building items in a finite element graph. Their relations completely describe the mesh and graph structures. Of interest to us are the adjacency structure and the dual adjacency structure, which give the adjacent relations in the graph and dual graph respectively. Different representations of the adjacency structure have been discussed in section 5.2. The connection table storage mode is most appropriate when the graph is regular† or near regular. By lemma 6.2, the dual graph is near-regular, since each triangular element has at most three neighboring

---

† A graph is regular if all its nodes have the same degree.

elements. It is then convenient and economical to store the dual graph using a connection table DUAL(|T|,3).

For each element T, it is sometimes useful to have a list of the nodes belonging to T. This collection over all elements may be appropriately called the element structure. Since the number of nodes associated with an element is usually fixed throughout the mesh, the element structure can be conveniently stored as a two dimensional array.

Another useful data structure for a finite element graph is a membership structure, which can be considered as the "inverse" relation of the element structure. For each node x, it has a list of all triangular elements to which x belongs. The structure is usually implemented in the form of linked lists or arrays similar to connection tables.

We now discuss the role played by these data representations in ordering finite element graphs. For the general purpose ordering algorithms in chapter 5, it is common to use the adjacency structure. In [George71], George implements the RCM scheme for finite element systems using the element and membership structures. This selection is more oriented to finite element applications. To retrieve the adjacent set of a node x, we first identify those triangular elements that contain x, using the membership structure. The neighbors, which are the nodes sharing common triangles with the node x, can then be found

from the element structure. Such a choice is most suitable for the IRCM algorithm.

The desirable data structures for a finite element graph in the EA algorithm are quite different. In producing an annihilation sequence, it requires the dual adjacency structure of the mesh. Then, to carry out the actual labelling of the nodes during triangle annihilation, the element structure is needed. Thus, the data structures involved in the EA algorithm are relatively simple.

To compare the storage requirements for the various forms of representation, we consider any given finite element mesh M(r,s,t). Let G=(X,E) be its associated finite element graph and $(\mathcal{T},\mathcal{E})$ be its dual graph. Define e to be 3r + 3s + t, the number of nodes in a triangular element. It is clear that the storage locations required for the various structures are:

Adjacency	$2\|E\| + \|X\|$,
Dual Adjacency	$3\|\mathcal{T}\|$,
Element	$e\|\mathcal{T}\|$,
Membership	$\geq e\|\mathcal{T}\|$.

To relate these quantities, let VB, VI, and H be the respective number of boundary vertices, interior vertices and holes in the given mesh M. We quote the following results.

Lemma 6.8: ([Ewing70])

$$|\mathcal{T}| = 2VI + VB + 2H - 2,$$

$$|\mathcal{E}| = 3VI + VB + 3H - 3. \qquad \square$$

Lemma 6.9: ([George71])

$$|X| = (r+3s+2t)VI + (r+2s+t)VB + (3s+2t)H - (3s+2t),$$

$$2|E| + |X| = (p+r^2)VI + \tfrac{1}{3}(p+e^2)VB + pH - p,$$

where $p = 2e^2 - 3(s+2r)^2$. $\square$

The results in lemmas 6.8 and 6.9 can be used to estimate the amount of storage required to solve a particular finite element problem. For large application problems, we typically have

$$VI >> VB,$$

so that

$$\frac{|X|}{|\mathcal{T}|} \approx \frac{r + 3s + 2t}{2}$$

$$\frac{2|E| + |X|}{(3+e)\,|\mathcal{T}|} \approx \frac{2e^2 + r^2 - 3(2r+s)^2}{2(3+e)}.$$

These ratios indicate an advantage of the element-oriented storage schemes when high order elements ($s+t>0$) are used. We tabulate the ratios in table 6.2 for different values of r, s, and t.

(r,s,t)	$\frac{\|X\|}{\|\Upsilon\|}$	$\frac{2\|E\| + \|X\|}{(3+e)\|\Upsilon\|}$
(1,0,0)	0.5	0.583
(1,1,0)	2.0	2.556
(1,2,1)	4.5	5.885

Table 6.2: Ratios of node against element storage.

In table 6.3, we compute the storage requirements for the various forms of representations on an actual finite element problem. These data correctly reflect the ratios given in table 6.2.

(r,s,t)	n	|X|	|E|	|T|	Adjacency Structure	Element + Membership Structures	Dual Adj + Element Structures
(1,2,1)	7	484	3612	98	7708	1960	1274
	8	625	4704	128	10033	2560	1664
	9	784	5940	162	12664	3240	2106
	10	961	7820	200	16601	4000	2600
(1,1,0)	12	625	3096	288	6817	3456	2592
	13	729	3627	338	7983	4056	3042
	14	841	4200	392	9241	4704	3528
	15	961	4815	450	10591	5400	4050
(1,0,0)	26	729	2080	1352	4889	8112	8112
	28	841	2408	1568	5657	9408	9408
	30	961	2760	1800	6481	10800	10800
	32	1089	3136	2048	7361	12288	12288

Table 6.3: Storage requirements for finite element graphs associated with the n by n regular right triangular mesh.

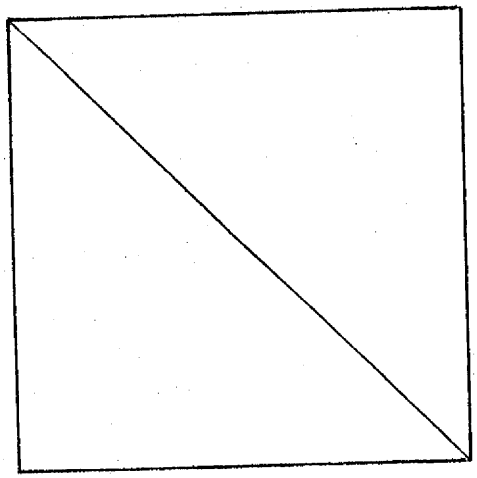
.5.2 Experimental Results

The EA algorithm is implemented in FORTRAN and a listing of the code can be found in the appendix. The program uses the dual adjacency and element structures to represent finite element graphs. The performance of this algorithm is compared with those ordering schemes studied in chapter 5, namely GPS, KING, LEVY, and RCM. In contrast, all these schemes utilize the adjacency structure.

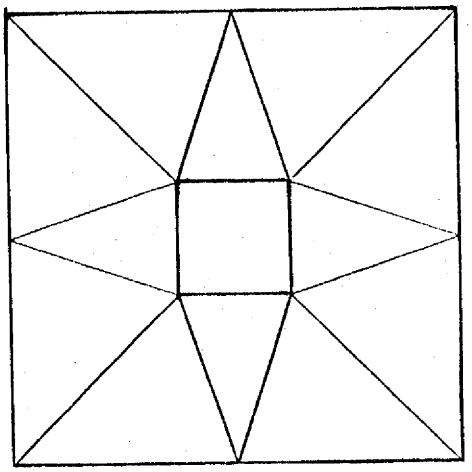
The test problems are provided by the current version of George's **mesh generation code**, which is an extension to the one given in [George71]. Input to the code are two parameters γ and μ, and a planar mesh M_0 which is a gross triangulation of the domain. The parameter γ is the subdividing factor, whereby each triangular side in the input mesh is evenly divided into γ segments by $\gamma-1$ nodes. By joining the new nodes in the obvious way, we obtain a refined mesh M having γ^2 times as many triangles as the original mesh. The second parameter μ governs the number and the distribution of nodes on the resulting mesh M. It corresponds essentially to the degree of certain piecewise polynomial bases used in finite element applications. The final element mesh generated is given by M(r,s,t), where

$$r = 1,\ s = \mu-1,\ \text{and}\ t = \tfrac{1}{2}(\mu-1)(\mu-2).$$

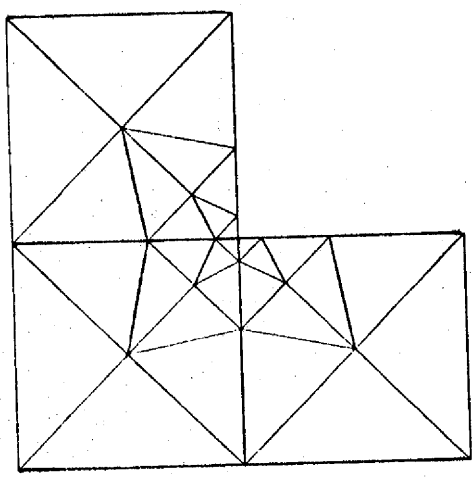
The set of test problems consists of six mesh problems designed by George. The basic meshes are shown in figure 6.10. For each mesh, we vary the values of γ and μ, and the experimental results are tabulated in tables 6.4-6.9.



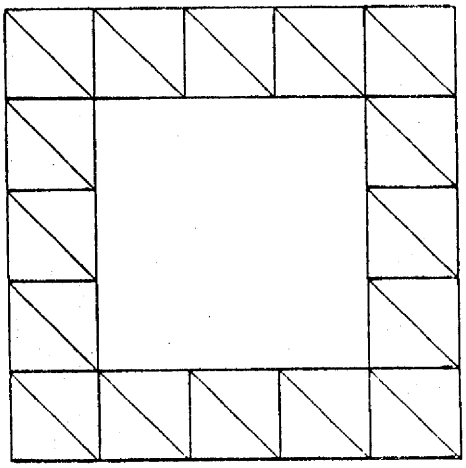
a) Square



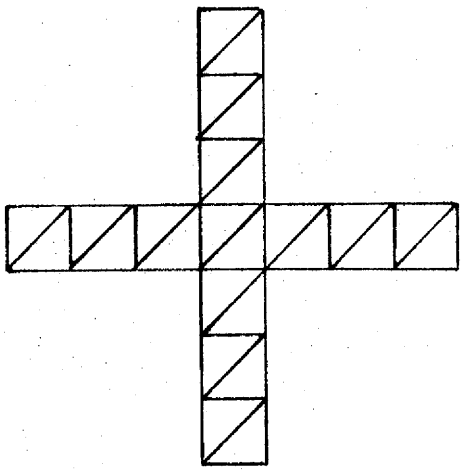
b) Hollow square (small hole)



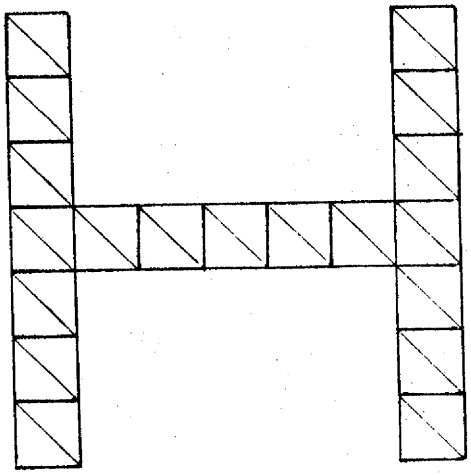
c) Graded L



d) Hollow square (large hole)



e) + shaped domain



f) H-shaped domain

Figure 6.10: Basic meshes for test problems.

μ	ν	|X|	|E|	|T|	ORDER TIME					PROFILE					OPERATION				
					EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM
3	7	484	3612	98	0.07	0.69	0.52	0.50	0.42	9057	18474	8985	10767	9936	106346	449660	104906	146351	127505
	8	625	4704	128	0.08	0.89	0.68	0.76	0.54	12984	27105	12894	16143	14157	167119	741370	165184	252628	197800
	9	784	5940	162	0.10	1.12	0.86	1.16	0.68	17892	38124	17784	22119	19404	250452	1162320	247968	371517	293040
	10	961	7320	200	0.11	1.44	1.11	1.72	0.87	23889	51669	23763	31365	25785	361286	1730558	358199	610061	418580
2	12	625	3096	288	0.14	0.59	0.49	1.13	0.33	11730	19180	11676	15392	12298	139153	367316	138154	229466	152079
	13	729	3627	338	0.17	0.69	0.58	1.49	0.39	14663	24113	14603	19617	15341	184947	495000	183777	317164	201292
	14	841	4200	392	0.20	0.81	0.68	1.95	0.45	18046	29812	17980	24446	18844	241153	652541	239800	424263	261473
	15	961	4815	450	0.21	0.94	0.78	2.50	0.50	21911	36349	21839	30079	22839	309235	845503	307687	558783	334114
1	26	729	2080	1352	0.67	0.51	0.46	2.10	0.23	13478	13455	13455	15755	13455	159628	159237	159237	208687	159237
	28	841	2408	1568	0.73	0.60	0.53	2.78	0.27	16671	16646	16646	19571	16646	209946	209496	209496	276771	209496
	30	961	2760	1800	0.86	0.71	0.61	3.59	0.30	20332	20305	20305	23959	20305	271298	270785	270785	360308	270785
	32	1089	3136	2048	1.03	0.82	0.70	4.58	0.33	24493	24464	24464	28959	24464	345188	344608	344608	461478	344608

Table 6.4: Square domain.

μ	γ	|X|	|E|	|T|	ORDER TIME EA	GPS	KING	LEVY	RCM	PROFILE EA	GPS	KING	LEVY	RCM	OPERATION EA	GPS	KING	LEVY	RCM
3	2	252	1800	48	0.05	0.45	0.35	0.19	0.31	3821	6983	4079	3872	4281	36286	121489	41386	37201	45699
	3	540	3996	108	0.08	0.93	0.77	0.64	0.63	11972	21705	12605	11867	12275	158374	521220	176872	154102	166483
	4	936	7056	192	0.11	1.63	1.38	1.56	1.08	27815	49221	28745	26342	26534	481780	1516209	519349	424930	436186
2	3	252	1188	108	0.07	0.26	0.23	0.23	0.18	3667	5130	3858	3553	3650	33882	64712	37714	31619	33542
	4	432	2088	192	0.10	0.43	0.40	0.57	0.28	8463	11387	8738	7962	7874	101355	178918	108929	88587	87566
	5	660	3240	300	0.15	0.69	0.62	1.22	0.42	16247	21283	16638	14891	14450	239156	402537	252816	197473	188240
	6	936	4644	432	0.21	0.95	0.90	2.33	0.59	27755	35672	28294	25334	23906	484957	786126	507687	396953	356872
1	9	540	1512	972	0.45	0.35	0.34	1.19	0.17	12184	9693	9540	9937	9693	167587	106756	103390	110060	106756
	10	660	1860	1200	0.57	0.43	0.41	1.75	0.20	16564	13027	12837	13357	13027	250709	155722	151162	160573	155722
	11	792	2244	1452	0.71	0.53	0.49	2.47	0.23	21886	17047	16816	17509	17047	361524	219794	213788	227299	219794
	12	936	2664	1728	0.87	0.64	0.58	3.41	0.27	28242	21817	21541	22423	21817	505600	301788	294060	312416	301788

Table 6.5: Square domain with small hole.

					ORDER TIME					PROFILE					OPERATION				
μ	γ	|X|	|E|	|T|	EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM
3	1	154	1116	30	0.05	0.36	0.22	0.12	0.19	1908	2913	1908	2604	2139	15657	36669	15657	28620	19410
	2	577	4392	120	0.08	1.40	1.19	0.89	1.04	14226	21549	13896	18867	14111	208239	498258	196140	373728	202672
	3	1270	**	270	0.15	**	**	**	**	47952	**	**	**	**	1038099	**	**	**	**
2	1	73	333	30	0.04	0.10	0.08	0.04	0.07	567	728	567	767	618	3256	5184	3256	5766	3782
	2	265	1296	120	0.08	0.32	0.27	0.28	0.21	4223	5102	4142	5470	3898	42409	62103	40995	71833	36413
	3	577	2889	270	0.14	0.70	0.63	1.07	0.45	14236	16301	13967	23922	11972	209612	278567	201916	628275	149679
	4	1009	5112	480	0.24	1.27	1.16	2.94	0.78	33892	37521	33529	60423	26967	661006	825303	643927	2127257	422357
1	5	406	1155	750	0.36	0.31	0.38	0.71	0.26	8529	6371	6570	7038	7449	109471	62975	67144	77080	84381
	6	577	1656	1080	0.55	0.44	0.57	1.35	0.38	14716	10763	11145	11999	12590	224338	123326	132689	153835	165659
	7	778	2247	1470	0.78	0.61	0.82	2.40	0.55	23353	16815	17465	18561	19676	412468	219135	237404	269020	294892
	8	1009	2928	1920	1.07	0.82	1.14	3.96	0.77	34837	24793	25812	27328	29019	699439	362124	394447	443392	487992

Table 6.6: L-shaped domain.

μ	γ	$\lvert X\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	ORDER TIME					PROFILE					OPERATION				
					EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM
3	1	192	1248	32	0.05	0.23	0.16	0.13	0.13	1868	3228	1868	1952	1971	12434	35218	12434	13460	13879
	2	672	4800	128	0.08	0.79	0.51	0.80	0.39	10502	20115	10871	13028	11193	101085	364571	108366	161313	115448
	3	1440	**	288	0.14	**	**	**	**	31439	**	**	**	**	**	**	**	**	**
2	1	96	384	32	0.04	0.10	0.08	0.05	0.07	609	864	609	630	628	2960	5576	2960	3146	3134
	2	320	1440	128	0.08	0.29	0.20	0.34	0.15	3285	4912	3401	4212	3412	22492	48186	24084	37786	24276
	3	672	3168	288	0.13	0.62	0.43	1.24	0.29	9668	14440	9725	12658	9746	87011	188615	88110	156530	88554
	4	1152	5568	512	0.21	1.11	0.74	3.50	0.47	21112	31791	21009	28670	21048	233225	518798	231216	459637	232144
1	5	480	1280	800	0.31	0.34	0.27	0.95	0.15	5417	5382	5242	5290	5382	40150	39702	37742	38296	39702
	6	672	1824	1152	0.46	0.50	0.38	1.80	0.19	8840	8779	8575	8660	8779	74103	73194	69930	71025	73194
	7	896	2464	1568	0.62	0.70	0.50	3.13	0.25	13453	13360	13080	13225	13360	125774	124200	119160	121248	124200
	8	1152	3200	2048	0.87	0.96	0.65	5.17	0.33	19432	19301	18933	19179	19301	200398	197920	190560	194226	197920

Table 6.7: Square domain with large hole.

μ	γ	$\lvert X\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	ORDER TIME					PROFILE					OPERATION				
					EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM	EA	GPS	KING	LEVY	RCM
3	1	160	1020	26	0.04	0.35	0.27	0.10	0.25	1380	1884	1656	1524	1713	8624	15608	12206	10664	13097
	2	553	3912	104	0.07	1.15	0.92	0.60	0.80	9237	11290	11004	9249	9856	101560	150567	144892	106381	113413
	3	1180	8676	234	0.14	2.47	2.06	2.25	1.73	30417	33870	33954	26472	28788	505626	619356	632502	399426	435807
2	1	81	315	26	0.05	0.11	0.09	0.04	0.08	447	603	549	519	554	2037	3396	2970	2757	3015
	2	265	1176	104	0.07	0.31	0.26	0.24	0.21	2915	3111	3486	2679	3029	22870	24566	32526	20063	24123
	3	553	2583	234	0.13	0.68	0.54	0.85	0.43	9428	9040	10523	8175	8717	109610	94033	136377	85966	90521
	4	945	4536	416	0.21	1.19	0.95	2.32	0.68	22039	19752	23524	19855	18917	342981	254761	390481	305688	241666
1	5	396	1045	650	0.31	0.30	0.26	0.66	0.16	5843	4332	4422	3602	4936	61775	31270	34556	23830	42790
	6	553	1488	936	0.44	0.44	0.36	1.24	0.21	9909	7054	7302	5828	8064	124616	57541	65715	43240	79592
	7	736	2009	1274	0.61	0.60	0.49	2.12	0.26	15338	10721	11220	8860	12288	220763	97500	114333	73130	136030
	8	945	2608	1664	0.88	0.80	0.61	3.44	0.33	22465	15473	16336	12758	17772	364316	155198	185977	115734	218057
	9	1180	3285	2106	1.11	1.07	0.79	5.34	0.42	31512	21450	22810	17678	24680	568470	235208	286988	175060	332412

Table 6.8: +-shaped domain.

μ	γ	|X|	|E|	|T|	ORDER TIME EA	GPS	KING	LEVY	RCM	PROFILE EA	GPS	KING	LEVY	RCM	OPERATION EA	GPS	KING	LEVY	RCM
3	1	232	1488	38	0.05	0.30	0.20	0.15	0.18	1944	2181	2040	1800	2127	11912	14735	12872	10472	14222
	2	805	5712	152	0.10	1.28	0.97	1.12	0.84	10926	16216	12432	10302	11875	87099	217881	124633	93118	114321
	3	1720	**	342	0.17	**	**	**	**	35730	**	**	**	**	463092	**	**	**	**
2	2	385	1716	152	0.09	0.41	0.30	0.44	0.24	3393	3660	3883	3142	3583	21482	24374	27626	19938	23693
	3	805	3771	342	0.16	0.88	0.64	1.66	0.47	10940	12806	11581	9044	10191	99060	137205	112202	74326	85875
	4	1377	6624	608	0.27	1.59	1.14	4.60	0.80	24735	27904	25667	19694	21938	286863	368784	315366	199838	223977
1	5	576	1525	950	0.39	0.48	0.34	1.32	0.20	6463	5682	5645	5286	5682	51428	39301	39268	40084	39301
	6	805	2172	1368	0.57	0.72	0.47	2.52	0.27	10776	9254	9275	8943	9254	99851	72307	73601	82132	72307
	7	1072	2933	1862	0.79	1.04	0.63	4.37	0.34	16621	14067	14198	12811	14067	175692	122526	126639	118492	122526

Table 6.9: H-shaped domain.

5.3 Comparative Remarks

Some general observations on the performances of the various algorithms will be drawn from the set of tabulated results.

Among algorithms GPS, KING, LEVY, and RCM, the RCM scheme requires the least amount of execution time. In some cases, it runs twice as fast as its strongest competitor. In general, with respect to the ordering time, the algorithms may be ordered as follows:

time(RCM) < time(KING) ≃ time(GPS) < time(LEVY).

The EA algorithm exhibits some interesting behavior in its execution time. The differences between its running time and those of the others for higher order element meshes are most striking. For mesh M(1,2,1), EA runs as much as 6 to 10 times faster than RCM. For M(1,1,0), it requires still 2 to 3 times less ordering time. The significant difference can be easily explained by the different magnitudes of the quantities $|X|$ and $|\Upsilon|$ (see table 6.2). The EA algorithm operates on the dual graph, which is much smaller in size than the corresponding finite element graphs for high order elements.

The same reasoning applies to the case M(1,0,0). Since $|\Upsilon| \simeq 2|X|$, it is not surprising to see that RCM runs 2 to 3 times faster than EA. However, the ordering time of EA is comparable to that of GPS and KING for M(1,0,0).

We now compare the performances of the algorithms as profile minimizers. For our set of test problems, algorithms KING, LEVY and RCM perform quite consistently. They produce profiles reasonably close to the lowest. It is interesting to point out that the LEVY scheme performs extremely well for meshes with appendages. Indeed, for the "+" and "H"-shaped domains, it produces the smallest profile in nearly all cases.

For M(1,0,0), GPS generates profiles comparable to that of KING, LEVY, and RCM. But its performance on higher order element meshes is less consistent. In many cases, its profile almost doubles the others.

In contrast, the EA algorithm performs reasonably well for higher order element meshes. Together with its small storage requirement (table 6.2) and its extremely fast running time, we feel that EA is a practical alternative when one is considering profile minimization of high order element meshes.

The performance of EA on M(1,0,0) is comparatively less effective. Since its performance is insensitive to the degree μ of interpolation (which is one of the advantages of this approach), it is apparent that the annihilation sequence given by the EA algorithm is far from the best for large meshes. Indeed, the tabulated data imply that the annihilation sequence induced by the IRCM algorithm (section 6.4.2) is often better in this respect. The

efficient determination of effective annihilation sequences from the dual graph structure is certainly a practical problem deserving future investigation.

Chapter 7 Concluding Remarks

In the in-core solution of a given linear system by the envelope method, the central problem is to determine a permutation matrix so that the correspondingly permuted system has a small profile. We have used a graph-theoretic approach to study the method and its related ordering problems. Based on the envelope structure of a graph, the minimum and minimal envelope problems have been posed.

Partial success has been achieved in tackling these problems for the class of N by N matrices associated with tree structures. We have presented an $O(N \log_2 N)$ recursive algorithm which always generates a minimal envelope ordering for trees of N nodes. While the minimal ordering problem is of graph-theoretic interest, our final goal is to solve the minimum problem. The design of an efficient algorithm to find minimum orderings for tree structures is itself an interesting research problem.

The ordering problem becomes much more difficult for general symmetric matrices. This can be reflected by the heuristic nature of all the existing practical algorithms. As illustrated by the examples in section 5.6, these heuristic algorithms may give very poor orderings. However, for application problems, they are typically $O(N^{\alpha})$ algorithms ($1 \leq \alpha \leq 2$), which generally produce profiles reasonably small for practical purposes.

We have studied and analysed the algorithms of Cuthill-McKee, King, Levy and reverse Cuhill-McKee. In particular, a graph-theoretic proof has been given to show that the reverse Cuthill-McKee algorithm can never be inferior to the original Cuthill-McKee scheme in the envelope context. Moreover, our experimental results are in favour of the reverse algorithm as a general purpose profile reduction scheme. Indeed, when a pseudo-peripheral starting node is used, the algorithm is extremely fast and consistently effective.

All the aforementioned algorithms have a common feature. They examine the graph structure only locally so as to reduce the envelope size, which is a global quantity. The insufficiency of local information suggests the possibility of devising algorithms that take the entire structure of the graph into account. In some sense, the algorithm by Gibbs, Poole, and Stockmeyer may be regarded as a global strategy for bandwidth reduction. Along the same line, Gibbs [Gibbs75] recently has developed one for profile reduction. The algorithm generates a "long and thin" global level structure and then imposes a King-like numbering on each level. It is felt that this approach of combining local and global strategies can be quite effective.

Finite element graphs constitute a special class of graphs, and they arise from the application of the finite element method. We have exploited the structure of these

graphs and have shown that the notion of element annihilation is an alternative approach for profile minimization. In that case, the crucial factor becomes the ordering on the set of triangular elements (annihilation sequence). This approach has a distinctive advantage. For a given finite element mesh M(r,s,t), the effectiveness of a particular annihilation sequence is independent of the parameters r, s, and t. Thus, the same sequence can be used for mesh problems with the same mesh structure M but different interpolating polynomials.

To apply the idea of element annihilation, we have developed an algorithm for producing reasonably good annihilation sequences for two dimensional domains with triangular elements. For high order element meshes, its economy in data information, its speed, and its effective performance show that the scheme is a practical alternative to other methods for reducing profiles. Like the node orderings, this algorithm is based on local information to number the triangular elements. Combinations of local and global strategies may result in better annihilation sequences. It is a possible avenue for future investigation.

Although we have restricted ourselves to two dimensional domains with triangular elements, it should be clear that the concept of element annihilation can be extended to regions with quadrilateral elements or to three dimensional domains. For two dimensional quadrilateral

meshes, we know [Ewing70]:

$$|\mathbb{E}| = \tfrac{1}{2}(VB + 2VI) + H - 1,$$

where $\mathbb{E}$ is the set of quadrilateral elements, and VB, VI and H are as before the respective number of boundary vertices, interior vertices and holes in the mesh. It follows then

$$|\mathbb{E}| \leq VB + VI.$$

This suggests a potential advantage of the element approach in this application since the number of elements can never exceed the number of nodes.

References

[Aho74] Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.
The Design and Analysis of Computer Algorithms;
Addison-Wesley, Reading, Mass. (1974).

[Akyuz68] Akyuz, F.A.; Utku, S.
"An automatic relabelling scheme for bandwidth minimization of stiffness matrices"; AIAA 6 (1968) 728-730.

[Alway65] Alway, G.G.; Martin, D.W.
"An algorithm for reducing the bandwidth of a matrix of symmetric configuration"; Comput. J. 8 (1965) 264-272.

[Arany71] Arany, I; Smyth, W.F.; Szoda, L.
"An improved method for reducing the bandwidth of sparse symmetric matrices"; Proceedings IFIP (1971).

[Bauer69] Bauer, H.R.; Becker, S.; Graham, S.L.; Satterthwaite, E.
"Algol W language description"; Computer Science Dept, Stanford University (1969).

[Berge58] Berge, C.
The Theory of Graphs and its Applications;
Wiley, New York(1958).

[Birkhoff73] Birkhoff, G.; George, J.A.;
"Elimination by nested dissection"; in Complexity of Sequential and Parallel Numerical Algorithms, edited by Traub, J.F.; Academic Press, New York and London (1973).

[Bolstad73] Bolstad, J.H.; Leaf, G.K.; Linderman, A.J.; Kaper, H.G.
"An empirical investigation of reordering and data management for finite element systems of equations"; ANL-8050 Argonne National Laboratory, Argonne, Illinois (1973).

[Bunch74] Bunch, J.R.; Rose, D.J.
"Partitioning, Tearing and Modification of Sparse Linear Systems"; J. Math. Anal. and Appl. 48 (1974) 574-593.

[Busacker65] Busacker, R.G.; Saaty, T.L.
Finite Graphs and Their Networks; McGraw-Hill, New York (1965).

[Chang69] Chang, A.
"Application of sparse matrix methods in electric power system analysis"; in Sparse Matrix Yorktown Conference Proceedings (1969).

[Cheng73a] Cheng, K.Y.
"Minimizing the bandwidth of sparse symmetric matrices"; Computing 11 (1973) 103-110.

[Christian73] Christian, M.
"Solution of linear equations --- state-of-the-art"; J. Structural Division ASCE 99 (1973) 1507-1525

[Cheng73b] Cheng, K.Y.
"Note on minimizing the bandwidth of sparse symmetric matrices"; Computing 11 (1973) 27-30.

[Churchill71] Churchill, M. "A sparse matrix procedure for power systems analysis programs"; in Large Sparse Sets of Linear Equations, edited by Reid, J.K.; Academic Press, London (1971).

[Collins73] Collins, R.J.
"Bandwidth reduction by automatic renumbering"; Intern. J. Numer. Methods in Engrg. 6 (1973) 345-356.

[Crane75] Crane, H.L.Jr.; Gibbs, N.E.; Poole, W.G.Jr.; Stockmeyer, P.K.
"Matrix bandwidth and profile minimization"; Report No. 75-9, ICASE Report (1975).

[Cuthill69] Cuthill, E.; McKee, J.
"Reducing the bandwidth of sparse symmetric matrices"; Proc. ACM 23rd National Conference (1969).

[Cuthill72] Cuthill, E.
"Several strategies for reducing the bandwidth of matrices"; in Sparse Matrices and their Applications, edited by Rose,D.J. and Willoughby,R.A.; Plenum Press, New York (1972).

[Eisenstat74] Eisenstat, S.C.; Sherman, A.H.
"Subroutines for envelope solution of sparse linear systems"; Research Report No. 35, Dept of Computer Science, Yale University (1974).

[Everstine72] Everstine, G.C.
"The BANDIT computer program for the reduction of matrix bandwidth for NASTRAN"; NSRDC Report 3827 (1972).

[Ewing70] Ewing, D.J.F.; Fawkes, A.J.; Griffiths, J.R.
"Rules governing the number of nodes and elements in a finite element mesh"; Intern. J. Numer. Method in Engrg. 2(1970) 597-600.

[Fuchs72] Fuchs, G.; Roy, J.R.; Schrem, E.
"Hypermatrix solution of large sets of symmetric positive definite linear systems"; Computer Methods in Applied Mechanics and Engineering 1(1972) 197-216.

[George71] George, J.A.
"Computer Implementation of the finite element method"; Stanford Computer Science Dept., Technical Report STAN-CS-71-208; Stanford, California (1971).

[George72] George, J.A.
"An efficient band-oriented scheme for solving n by n grid problems"; Proc. Fall Joint Computer Conference (1972) 1317-1320.

[George73a] George, J.A.
"Nested dissection of a regular finite element mesh"; SIAM J. Numer. Anal. 10 (1973) 345-363.

[George73b] George, J.A.
"A survey of sparse matrix methods in the direct solution of finite element equations"; Proc. Summer Computer Simulation Conference, Montreal, Canada (1973) 15-20.

[George74] George, J.A.
"On block elimination for large sparse linear systems"; SIAM J. Numer. Anal. 11 (1974) 585-603.

[George75a] George, J.A.
"Numerical experiments using dissection methods to solve n by n grid problems"; Research Report CS-75-07, Dept of Computer Science, University of Waterloo (1975).

[George75b] George, J.A.; Liu, J.W.H.
"A note on fill for sparse matrices"; SIAM J. Numer. Anal. 12 (1975) 452-455

[George75c] George, J.A.; Liu, J.W.H.
"An automatic partitioning and solution scheme for solving large sparse positive definite systems of linear algebraic equations"; Research Report CS-75-17, Dept of Computer Science, University of Waterloo (1975).

[Gibbs74a] Gibbs, N.E.; Poole, W.G.; Stockmeyer, P.K.
"An algorithm for reducing the bandwidth and profile of a sparse matrix"; ICASE Report (1974); to appear in SIAM J. Numer. Anal.

[Gibbs74b] Gibbs, N.E.; Poole, W.G.; Stockmeyer, P.K.
"A comparison of several bandwidth and profile reduction algorithms"; ICASE Report (1974).

[Gibbs75] Gibbs, N.E.
"A hybrid profile reduction algorithm"; manuscript (1975).

[Gustavson70] Gustavson, F.D.; Liniger, W.M.; Willoughby, R.A.
"Symbolic gerenation of an optimal Crout algorithm for sparse systems of linear equations"; J. Assoc. Comput. Mach. 17 (1970) 87-109.

[Gustavson72] Gustavson, F.G.
"Some basic techniques for solving sparse systems of equations"; in Sparse Matrices and their Applications, edited by Rose, D.J. and Willoughby, R.A.; Plenum Press, New York (1972).

[Hoffman73] Hoffman, A.J.; Martin, M.S.; Rose, D.J.
"Complexity bounds for regular finite difference and finite element grids"; SIAM J. Numer. Anal. 10 (1973) 364-369

[Hussey70] Hussey, M.J.L; Thatcher, R.W.; Bernal, M.J.M.
"Construction and use of finite elements"; J. Inst. Math. Appl. 6 (1970) 262-283.

[Irons70] Irons, B.M.
"A frontal solution program for finite element analysis"; Intern. J. Numer. Method in Engrg. 2 (1970) 5-32.

[Jennings66] Jennings, A.
"A compact storage scheme for the solution of symmetric simultaneous equations"; Comput. J. 9 (1966) 281-285.

[Jennings68] Jennings, A.
"A sparse matrix scheme for the computer analysis of structures"; Intern. J. Comput. Math. 2 (1968) 1-21.

[King70] King, I.P.
"An automatic reordering scheme for simultaneous equations derived from network systems"; Intern. J. Numer. Method in Engrg. 2 (1970) 523-533.

[Knuth68] Knuth, D.E.
Fundamental Algorithms; Addison-Wiley, Reading, Mass. (1968).

[Larcombe71] Larcombe, M.
"A list processing approach to the solution of large sparse sets of matrix equations and the factorization of the overall matrix"; in Large Sparse Sets of Linear Equations, edited by Reid, J.K.; Academic Press, London (1971).

[Lee69] Lee, H.
"An implementation of Gaussian elimination for sparse systems of linear equations"; in Sparse Matrix Yorktown Conference Proceedings (1969).

[Levy71] Levy, R.
"Resequencing of the structural stiffness matrix to improve computational efficiency"; JPL Quarterly Tech. Review 1 (1971) 61-70.

[Liu75] Liu, J.W.H.; Sherman, A.H.
"Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices"; to appear in SIAM J. Numer. Anal.

[Martin65] Martin, R.S.; Wilkinson, J.H.
"Symmetric decomposition of positive definite band matrices"; Numer. Math. 7 (1965) 355-361.

[Mayeda72] Mayeda, W.
Graph Theory; Wiley, Interscience (1972).

[Melosh69] Melosh, R.J.; Bamford, R.M.
"Efficient solution of load deflection equations"; J. Struct. Div. ASCE, (1969) 661-676.

[Ogbuobiri70] Ogbuobiri, E.C.; Tinney, W.F.; Walker, J.W.
"Sparsity-directed decomposition for Gaussian elimination on matrices"; IEEE Trans on Power Apparatus and Systems 89 (1970) 141-150.

[Ohtsuki75a] Ohtsuki, T.; Cheung, L.K.; Fujisawa, T.
"Minimal triangulation of a graph and optimal pivoting order in a sparse matrix", to appear in J. Math Anal. and Appl.

[Ohtsuki75b] Ohtsuki, T.
"A fast algorithm for finding an optimal ordering in the vertex elimination on a graph"; submitted to SIAM J. Computing.

[Parter61] Parter, S.V.
"The use of linear graphs in Gauss elimination"; SIAM Review 3 (1961) 119-130.

[Pooch73] Pooch, U.W.; Nieder, A.
"A survey of indexing techniques for sparse matrices"; ACM Computing Survey (1973) 109-133.

Rheinboldt73] Rheinboldt, W.C.; Mestztenyi, C.K.
"Programs for the solution of large sparse matrix problems based on the arc-graph structure"; Technical Report TR-262 (1973), Computer Science Centre, University of Maryland.

[Rose70] Rose, D.J.
"Triangulated graphs and the elimination process"; J. Math. Anal. and Appl. 32 (1970) 597-609.

[Rose72a] Rose, D.J.
"A graph-theoretic study of numerical solution sparse positive definite system of linear equations"; in Graph Theory and Computing, edited by Read, R.C.; Academic Press (1972).

[Rose72b] Rose, D.J.; Bunch, J.R.
"The role of partitioning in the solution of sparse linear systems"; in Sparse Matrices and their Applications, edited by Rose, D.J. and Willoughby, R.A.; Plenum Press, New York (1972).

[Rose75] Rose, D.J.; Tarjan, R.E.
"Algorithmic aspects of vertex elimination on graphs"; to appear in SIAM J. Computing.

[Rosen68] Rosen, R.
"Matrix bandwidth minimization"; Proc. ACM 23rd Computer Conference (1968).

[Segethova70] Segethova, J.
"Elimination procedures for sparse symmetric linear systems of a special structure"; Tech. Rep. 70-121, Computer Science Center, University of Maryland (1970).

[Strang73] Strang, G.; Fix, G.J.
An Analysis of the Finite Element Analysis; Prentice-Hall, Inc., Englewood Cliffs, N.J. (1973).

[Symposium75] Symposium on Sparse Matrix Computations; Argonne National Laboratory (1975).

[Tarjan72] Tarjan, R.E.
"Depth first search and linear graph algorithms"; SIAM J. Computing 1 (1972) 146-160.

[Tewarson73] Tewarson, R.P.
Sparse Matrices; Academic Press, New York (1973).

[Tinney69] Tinney, W.F.
"Comments on using sparsity techniques for power system problems"; in Sparse Matrix Yorktown Conference Proceedings (1969) 25-34.

[Tutte67] Tutte, W.T.
The Connectivity of Graphs; Toronto University Press, Toronto (1967).

[Varga62] Varga, R.S.
Matrix Iterative Analysis; Prentice-Hall, Englewood Cliffs, N.J. (1962).

[Wang73] Wang, P.T.R.
Bandwidth minimization, reducibility, decomposition, and triangularization of sparse matrices; Ph. D. dissertation, Dept of Computer and Information Science, Ohio State University (1973).

[Westlake69] Westlake, J.R.
A Handbook of Numerical Matrix Inversion and Solution of Linear Equations; Wiley, New York (1969).

[Wilkinson65] Wilkinson, J.H.
The Algebraic Eigenvalue Problem; Clarendon Press, London (1965).

[Willoughby69] Willoughby, R.A. (editor)
Proceeding of the Symposium on Sparse Matrices and their Applications (1969).

[Willoughby72] Willoughby, R.A.
"A survey of sparse matrix technology"; Rep. RC 3872, IBM, Yorktown Heights, N.Y. (1972).

[Wilson74] Wilson, E.L.; Bathe, K.J.; Doherty, W.P. "Direct solution of large systems of linear equations"; Computers and Structures 4 (1974) 363-372.

Zienkiewicz70] Zienkiewicz, O.C. The Finite Element Method in Engineering Science; McGraw Hill, London (1970).

Appendix

The various ordering algorithms discussed in the thesis are implemented in FORTRAN. A listing and description of the programs are given in this appendix.

Graphs are always assumed to be represented in the form of connection tables (section 5.2). The following variables are used to describe a graph G=(X,E):

N ---

is the number |X| of nodes in the graph G.

NR ---

is the maximum degree of the nodes, and is the row dimension of the connection table ADJNCY.

ADJNCY(NR,N) ---

is the connection table containing the adjacency structure of the graph. ADJNCY(j,i) is the j-th neighbor of node i.

DEG(N) ---

is an integer array which stores the degree of the nodes. The neighbors of node i can be retrieved as: ADJNCY(1,i), ... , ADJNCY(DEG(i),i).

MASK(N) ---

is an integer array with 0's and 1's, used to mask off subgraphs. It is introduced so that our ordering implementations can be general enough to be applied to subgraphs.

For clarity, the collection of subroutines is subdivided according to their functions.

A.1 Utility Routines

These routines perform certain basic operations as required by the various ordering schemes. Included in this set are "COPY", "RCOPY", "MINDEG", "SORT", and "FNDNBR". Their functions are self-explanatory.

```
C*************************************************************
C
C     COPY COPIES THE N1 ELEMENTS FROM ARRAY1 TO ARRAY2.
C
C*************************************************************
C
      SUBROUTINE COPY(N1, ARRAY1, ARRAY2)
         INTEGER ARRAY1(1), ARRAY2(1)
C
         IF (N1 .LE. 0) RETURN
         DO 100 I=1,N1
100         ARRAY2(I) = ARRAY1(I)
         RETURN
      END
C
C*************************************************************
C
C     RCOPY COPIES THE N1 ELEMENTS FROM ARRAY1 TO ARRAY2
C     FROM N1 TILL 1. THIS AVOIDS OVERRIDING PROBLEM WHEN
C     ARRAY1 AND ARRAY2 ARE THE SAME VECTOR.
C
C*************************************************************
C
      SUBROUTINE RCOPY(N1, ARRAY1, ARRAY2)
         INTEGER ARRAY1(1), ARRAY2(1)
C
         IF (N1 .LE. 0) RETURN
         IP = N1
         DO 100 I= 1,N1
            ARRAY2(IP) = ARRAY1(IP)
            IP = IP - 1
100      CONTINUE
C
         RETURN
      END
C
C*************************************************************
C
C     SUBROUTINE MINDEG RETURNS THE FIRST ENCOUNTERED MASKED
C     NODE OF MINIMUM DEGREE.  IT LOOKS FROM 'ISTART' TILL 'N'.
C
C*************************************************************
C
      INTEGER FUNCTION MINDEG(ISTART, N, MASK, DEG)
         INTEGER MASK(1), DEG(1)
C
         MIN = DEG(N)
         DO 100 I= ISTART, N
            IF ( (MASK(I) .EQ. 0) .OR.
     *           (DEG(I) .GE. MIN) ) GO TO 100
            MINDEG = I
            MIN = DEG(I)
100      CONTINUE
C
         RETURN
      END
```

```
C*************************************************************
C
C     SORT USES LINEAR INSERTION TO SORT THE NODES IN
C     ARRAY(1), . . ., ARRAY(NA),  IN INCREASING ORDER OF
C     THEIR VALUES IN 'F'.
C
C*************************************************************
C
      SUBROUTINE SORT(NA, ARRAY, F)
         INTEGER ARRAY(1), F(1)
C
         IF (NA .LE. 1) RETURN
         DO 300 K=2, NA
            NODE = ARRAY(K)
            L = K - 1
C
100         IF (L .LT. 1) GOTO 200
            IF (F(ARRAY(L)) .LE. F(NODE)) GOTO 200
            ARRAY(L+1) = ARRAY(L)
            L = L - 1
            GO TO 100
C
200         ARRAY(L+1) = NODE
300      CONTINUE
         RETURN
      END
C
C*************************************************************
C
C     FNDNBR FINDS ALL THE MASKED NEIGHBORS OF 'NODE' AND PUTS
C     THEM IN  ARRAY(I+1), . . ., ARRAY(I+T).  ON RETURN, 'I'
C     IS RESET TO THE VALUE  I+T.
C
C*************************************************************
C
      SUBROUTINE FNDNBR(NODE, ADJNCY, NR, DEG, MASK, ARRAY, I)
         INTEGER ADJNCY(NR,1), DEG(1), MASK(1), ARRAY(1)
         INTEGER NODE, I, NDEG
C
         NDEG = DEG(NODE)
         IF (NDEG .LE. 0) RETURN
C
         DO 100 J=1, NDEG
            NBR = ADJNCY(J, NODE)
            IF (MASK(NBR) .NE. 1) GO TO 100
C
            I = I + 1
            ARRAY(I) = NBR
            MASK(NBR) = 0
100      CONTINUE
C
         RETURN
      END
```

A.2 Starting Node Routines

This set consists of two subroutines, used for finding a pseudo-peripheral node. The level structure is a primary construct in the routines; the variables associated with it are described as follows.

NLVL ---

is the number of levels in the level structure, and it is equal to its length plus 1.

WIDTH ---

is the width of the level structure.

LS(N) ---

is the main storage for the level structure. Segments of this array correspond to levels of the structure.

XLS(*) ---

is the index array to the main vector LS(*). It is an integer array of length at least NLVL+1. The i-th level in the rooted level structure is given by:

LS(XLS(i)), ... , LS(XLS(i+1)-1).

The subroutines are explained in fair detail, so that they can be used effectively and modified easily. To get a pseudo-peripheral node, "FNROOT" should be called.

** ROOTLS(ROOT, ADJNCY, NR, DEG, MASK, LS, XLS, NLVL, WIDTH, IBORT)

Purpose: The subroutine generates the rooted level structure at a given input node. Only those masked nodes will be considered in the level structure.

Parameters:

ROOT --- is the input root at which level structure is to be generated.

ADJNCY, NR, DEG, MASK --- are the input variables describing the graph.

LS, XLS, NLVL, WIDTH --- are the output variables describing the rooted structure.

IBORT --- is an input parameter which triggers early return if WIDTH is greater than or equal to IBORT. In that case, the returned rooted level structure may only be partially formed.

Subroutines used: none.

** FNROOT(ROOT, ADJNCY, NR, N, DEG, MASK, LS, XLS, NLVL, LASTLV)

Purpose: This subroutine uses the finite iterative scheme by Gibbs, Poole and Stockmeyer to determine a pseudo-peripheral node for the masked component defined by a given node.

Parameters:

ROOT --- On input, it defines the component where a starting node is required. On return, it becomes the pseudo-peripheral node found.

ADJNCY, NR, N, DEG, MASK --- are the input variables describing the graph.

LS, XLS, NLVL --- are the working variables used to construct rooted level structures.

LASTLV(*) --- is a working array required by the algorithm to store the set of nodes in the last level of the previous level structure.

Subroutines used:

"ROOTLS", "COPY", "SORT".

```
C*************************************************************
C
C     ROOTLS GENERATES THE ROOTED LEVEL STRUCTURE AT THE NODE
C     'ROOT'.  ONLY THOSE MASKED NODES WILL BE CONSIDERED.
C
C*************************************************************
C
      SUBROUTINE ROOTLS( ROOT, ADJNCY, NR, DEG, MASK,
     *                   LS, XLS, NLVL, WIDTH, IBORT)
      INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *        LS(1), XLS(1)
      INTEGER WIDTH, LNBR, LBEGIN, LVLEND, ROOT
C
C     INITIALIZATION ....
C
      MASK(ROOT) = 0
      LS(1) = ROOT
C
      NLVL = 0
      WIDTH = 1
      LVLEND = 0
      LNBR = 1
C
C     'LBEGIN' IS THE POINTER TO THE BEGINNING OF PRESENT
C     LEVEL; AND 'LVLEND' POINTS TO THE END OF THIS LEVEL.
C
200   LBEGIN = LVLEND + 1
      LVLEND = LNBR
      NLVL = NLVL + 1
      XLS(NLVL) = LBEGIN
C
C     GENERATE THE NEXT LEVEL BY FINDING ALL THE MASKED
C     NEIGHBORS OF NODES IN PRESENT LEVEL.
C
      DO 400 I=LBEGIN, LVLEND
C
C        CALL FNDNBR(LS(I), ADJNCY, NR, DEG, MASK, LS, LNBR)
C        THE CODE FOR 'FNDNBR' IS INSERTED TO MINIMIZE
C        THE OVERHEAD IN SUBROUTINE CALLS.
C
         NODE = LS(I)
         NDEG = DEG(NODE)
         IF (NDEG .LE. 0) GO TO 400
C
         DO 300 J= 1,NDEG
            NBR = ADJNCY(J,NODE)
            IF (MASK(NBR) .NE. 1) GO TO 300
            LNBR = LNBR + 1
            LS(LNBR) = NBR
            MASK(NBR) = 0
300      CONTINUE
400   CONTINUE
C
C     COMPUTE THE CURRENT LEVEL WIDTH.
C     IF IT IS NONZERO AND LESS THAN "IBORT", BRANCH
C     BACK TO GENERATE NEXT LEVEL, OTHERWISE RETURN.
C
      LVSIZE = LNBR - LVLEND
      IF (LVSIZE .LE. 0) GO TO 500
      IF (LVSIZE .GT. WIDTH)  WIDTH = LVSIZE
      IF (WIDTH .LT. IBORT) GO TO 200
C
C     BEFORE RETURN, RESET ALL THE MASK OF THE NODES IN
C     THIS LS TO 1.
C
500   DO 600 I=1, LNBR
         MASK(LS(I)) = 1
600   CONTINUE
C
      XLS(NLVL+1) = LVLEND + 1
      RETURN
```

```
C*************************************************************
C
C     SUBROUTINE FNROOT DETERMINES A PSEUDO PERIPHERAL NODE
C     FOR THE 'MASK'-ED COMPONENT CONTAINING THE NODE 'ROOT'.
C     THE ITERATIVE SCHEME BY GIBBS ET. AL IS USED.
C
C*************************************************************
C
      SUBROUTINE FNROOT( ROOT, ADJNCY, NR, N, DEG, MASK,
     *                   LS, XLS, NLVL, LASTLV)
      INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *        LS(1), XLS(1), LASTLV(1)
      INTEGER ROOT, WIDTH
C
C     DETERMINE ROOTED LEVEL STRUCTURE AT 'ROOT'.
C
      CALL ROOTLS( ROOT, ADJNCY, NR, DEG, MASK,
     *             LS, XLS, NLVL, WIDTH, N)
      IF (NLVL .EQ. 1) RETURN
C
C     COPY THE LAST LEVEL OF LS TO VECTOR 'LASTLV'.
C     THEN SORT THE NODES BY DEGREE.
C
100   LLSIZE = XLS(NLVL+1) - XLS(NLVL)
      CALL COPY( LLSIZE, LS(XLS(NLVL)), LASTLV(1))
      CALL SORT( LLSIZE, LASTLV(1), DEG(1))
C
C     FOR EACH NODE IN THE LAST LEVEL, GENERATE ITS ROOTED
C     LEVEL STRUCTURE.  COMPARE ITS WIDTH AND LENGTH WITH
C     THE PREVIOUS LS.
C
      DO 400 I=1, LLSIZE
         NODE = LASTLV(I)
         CALL ROOTLS( NODE, ADJNCY, NR, DEG, MASK,
     *                LS, XLS, NNLVL, NWIDTH, WIDTH)
C
         IF (NWIDTH .GE. WIDTH) GO TO 400
         ROOT = NODE
         WIDTH = NWIDTH
C
         IF (NNLVL .LE. NLVL) GO TO 400
         NLVL = NNLVL
         ROOT = NODE
         GO TO 100
400   CONTINUE
C
      RETURN
      END
```

3 CM/RCM Routines

This set of routines is used to determine the Cuthill-McKee "GENCM" and reverse Cuthill-McKee "GENRCM" orderings.

** CM(ROOT,ADJNCY,NR,DEG,MASK,CCSIZE,ORDER)

Purpose: The subroutine determines the Cuthill-McKee ordering for a masked connected component.

Parameters:

ROOT --- is the starting node for the CM ordering.

ADJNCY, NR, DEG, MASK --- are the input variables describing the graph. On return, numbered nodes have their mask values switched to 0.

CCSIZE --- is an input variable containing the size of the connected component.

ORDER(*) --- is the output array containing the CM ordering. Here, ORDER(i) is the node numbered i in the ordering.

Subroutines used:

"FNBNBR", "SORT".

** GENCM(ADJNCY,NR,N,DEG,MASK,LS,XLS,LASTLV,PERM)

Purpose: The subroutine finds the CM ordering for a general (possibly disconnected) graph.

Parameters:

ADJNCY, NR, N, DEG, MASK --- are the input graph parameters. On return, the MASK values of numbered nodes become 0.

LS, XLS, LASTLV --- are the working arrays for level structures. They are used by "FNROOT".

PERM(N) --- is the output array containing the general CM ordering. PERM(i) is the new label for node i in the new ordering.

Subroutines used:

"MINDEG", "FNROOT", "CM".

** GENRCM(ADJNCY,NR,N,DEG,MASK,LS,XLS,LASTLV,PERM)

Purpose: This subroutine finds the reverse CM ordering for a general (possibly disconnected) graph.

Parameters: same as "GENCM", except that PERM becomes the RCM ordering.

Subroutines used: same as "GENCM".

```
C*****************************************************************
C
C     CM NUMBERS THE MASKED COMPONENT THAT CONTAINS 'ROOT'
C     USING THE CUTHILL-MCKEE ALGORITHM.
C     THE NUMBERING IS STARTED AT 'ROOT'.
C
C*****************************************************************
C
      SUBROUTINE CM( ROOT, ADJNCY, NR, DEG, MASK, CCSIZE, ORDER)
         INTEGER ADJNCY(NR,1), DEG(1), MASK(1), ORDER(1)
         INTEGER ROOT, PTR, FNBR, LNBR, CCSIZE
C
C        INITIALIZATION ....
C
         ORDER(1) = ROOT
         MASK(ROOT) = 0
C
         LVLEND = 0
         LNBR = 1
C
C        'LBEGIN' AND 'LVLEND' POINT TO THE BEGINNING AND THE
C        END OF THE CURRENT LEVEL RESPECTIVELY.
C
200      LBEGIN = LVLEND + 1
         LVLEND = LNBR
         IF (LVLEND .GE. CCSIZE) RETURN
C
C        FOR EACH NODE IN CURRENT LEVEL, FIND ITS NBRS IN THE
C        NEXT LEVEL.  SORT THEM BY DEGREE AND NUMBER THEM.
C        'FNBR' AND 'LNBR' POINT TO THE FIRST AND LAST UNNUMBERED
C        NBRS RESP OF THE CURRENT 'NODE' IN ARRAY 'ORDER'.
C
         DO 400 I=LBEGIN, LVLEND
            FNBR = LNBR + 1
            CALL FNDNBR(ORDER(I), ADJNCY, NR, DEG, MASK, ORDER, LNBR)
            IF (FNBR .LT. LNBR)
     *          CALL SORT(LNBR-FNBR+1, ORDER(FNBR), DEG)
400      CONTINUE
C
         IF (LNBR .GT. LVLEND) GO TO 200
         RETURN
       END
```

```
C*****************************************************************
C
C     SUBROUTINE GENRCM FINDS THE RCM RORDERING FOR A GENERAL
C     GRAPH.   FOR EACH CONNECTED COMPONENT IN THE GRAPH,
C     GENRCM FINDS THE CUTHILL-MCKEE ORDERING BY THE ROUTINE
C     'CM' AND THEN REVERSES IT.
C
C*****************************************************************
C
      SUBROUTINE GENRCM(ADJNCY, NR, N, DEG, MASK,
     *                  LS, XLS, LASTLV, PERM)
     *    INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *            LS(1), XLS(1), LASTLV(1), PERM(1)
      INTEGER ROOT, CCSIZE
C
C     TURN 'MASK' OF ALL NODES TO 1.
C
      DO 100 I=1, N
         MASK(I) = 1
100   CONTINUE
C
      NUM = 1
C
C     DETERMINE A MASKED NODE 'ROOT' OF MIN DEGREE.
C
      DO 400 I=1, N
         IF (MASK(I) .NE. 1) GO TO 400
         ROOT = MINDEG(I, N, MASK, DEG)
C
C        FIRST FIND A PSEUDO PERIPHERIAL NODE.  'CCSIZE' DENOTES
C        THE SIZE OF THE PRESENT CONNECTED COMPONENT.
C
         CALL FNROOT(ROOT, ADJNCY, NR, N, DEG, MASK,
     *               LS, XLS, NLVL, LASTLV)
         CCSIZE = XLS(NLVL+1) - 1
C
C        CM IS CALLED TO ORDER THE COMPONENT CONTAINING
C        THE NODE 'ROOT'.
C
         CALL CM(ROOT, ADJNCY, NR, DEG, MASK, CCSIZE, LS)
C
C        REVERSE THE NUMBERING OF THE NODES IN THE CM ORDERING.
C
         NUM = NUM + CCSIZE
         DO 300 J= 1,CCSIZE
            PERM(LS(J)) = NUM - J
300      CONTINUE
C
         IF (NUM .GT. N) RETURN
400   CONTINUE
      RETURN
      END
```

```
C*****************************************************************
C
C     SUBROUTINE GENCM FINDS THE CM ORDERING FOR A GENERAL
C     GRAPH.  FOR EACH CONNECTED COMPONENT, GENCM FINDS THE
C     ORDERING USING THE ROUTINE 'CM'.
C
C*****************************************************************
C
      SUBROUTINE GENCM(ADJNCY, NR, N, DEG, MASK,
     *                 LS, XLS, LASTLV, PERM)
     *    INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *            LS(1), XLS(1), LASTLV(1), PERM(1)
      INTEGER ROOT, CCSIZE
C
C     TURN 'MASK' OF ALL NODES TO 1.
C
      DO 100 I=1, N
         MASK(I) = 1
100   CONTINUE
C
      NUM = 1
C
C     DETERMINE A MASKED NODE 'ROOT' OF MIN DEGREE.
C
      DO 400 I=1, N
         IF (MASK(I) .NE. 1) GO TO 400
         ROOT = MINDEG(I, N, MASK, DEG)
C
C        FIRST FIND A PSEUDO PERIPHERIAL NODE.  'NSIZE' DENOT
C        THE SIZE OF THE PRESENT CONNECTED COMPONENT.
C
         CALL FNROOT(ROOT, ADJNCY, NR, N, DEG, MASK,
     *               LS, XLS, NLVL, LASTLV)
         CCSIZE = XLS(NLVL+1) - 1
C
C        CM IS CALLED TO ORDER THE COMPONENT CONTAINING
C        THE NODE 'ROOT'.
C
         CALL CM(ROOT, ADJNCY, NR, DEG, MASK, CCSIZE, LS)
C
         DO 300 J= 1,CCSIZE
            PERM(LS(J)) = NUM
            NUM = NUM + 1
300      CONTINUE
C
         IF (NUM .GT. N) RETURN
400   CONTINUE
      RETURN
      END
```

.4 KING/LEVY Routines

This set implements the King "GENKNG" and Levy "LEVY" algorithms for small profile orderings. "SUBDEG" is a utility routine used by both schemes and is hence included in this collection. Its function is clear from the documentation.

** KING(ROOT,ADJNCY,NR,DEG,MASK,CCSIZE,ORDER,FDEG)

Purpose: This subroutine determines the King ordering for a masked connected component.

Parameters:

ROOT --- is the starting node for the King ordering.

ADJNCY, NR, DEG, MASK --- are the input graph parameters. On return, numbered nodes have their mask values changed to 0.

CCSIZE --- is an input parameter containing the size of the connected component.

ORDER(*) --- is the resulting King ordering. ORDER(i) is the old node numbered i in the ordering.

FDEG(N) --- is a working vector used to store the current front degree of the nodes.

Subroutines used:

"SUBDEG", "RCOPY", "FNDNBR".

** GENKNG(ADJNCY,NR,N,DEG,MASK,LS,XLS,LASTLV,FDEG,PERM)

Purpose: This subroutine finds the King algorithm for a general (possibly disconnected) graph.

Parameters:

ADJNCY, NR, N, DEG, MASK --- are the input graph parameters.

LS, XLS, LASTLV --- are the working vectors for building level structures. They are used in "FNROOT".

FDEG(N) --- is a working vector used by "KING" to store current front degrees of the nodes.

PERM(N) --- is the output vector containing the general King ordering. PERM(i) is the new label assigned to node i in the ordering.

Surbroutines used.:

"MINDEG", "FNROOT", "KING".

** LEVY(ADJNCY, NR, N, DEG, MASK, ORDER, FDEG, PERM)

Purpose: The subroutine finds the Levy ordering for an input graph.

Parameters:

ADJNCY, NR, N, DEG, MASK --- are the input graph parameters. On return, the MASK values of numbered nodes become 0.

ORDER(N) --- is a working vector used to store the inverse of the permutation vector PERM.

FDEG(N) --- is a working vector used to store the current front degree of the nodes.

PERM(N) --- is the output permutation vector containing the LEVY ordering. PERM(i) is the new label for node i in the new ordering.

Subroutines used:

"SUBDEG".

```
C*****************************************************************
C
C     SUBROUTINE KING FINDS THE KING ORDERING FOR THE
C     CONNECTED 'MASK'-ED COMPONENT WITH 'ROOT'.
C     THE ORDERING STARTS AT THE NODE 'ROOT'.
C
C*****************************************************************
C
      SUBROUTINE KING( ROOT, ADJNCY, NR, DEG, MASK, CCSIZE,
     *                 ORDER, FDEG)
        INTEGER ADJNCY(NR,1), DEG(1), MASK(1), ORDER(1), FDEG(1)
        INTEGER ROOT, CCSIZE, FRBEG, FREND, FNBR, LNBR
C
C       INITIALIZATION
C       'FRBEG' AND 'FREND' POINT TO THE BEGINNING AND THE END
C       OF THE CURRENT FRONT RESPECTIVELY.
C
        FRBEG = 0
        FREND = 0
        LNBR = 1
C
C       PUT 'ROOT' INTO THE FRONT.
C
        ORDER(1) = ROOT
        MASK(ROOT) = 0
C
C       FOR EACH NEW NODE IN THE FRONT, SUBTRACT ONE FROM
C       'FDEG' OF ITS NEIGHBORS (USING ROUTINE 'SUBDEG').
C
100     FNBR = FREND + 1
        DO 200 J= FNBR, LNBR
           CALL SUBDEG( ORDER(J), ADJNCY, NR, DEG, FDEG)
200     CONTINUE
C
C       FROM THE FRONT SELECT THE NODE WITH MINIMUM INCREASE
C       IN FRONTWIDTH, I.E. WITH MINIMUM FDEG.
C
300     FRBEG = FRBEG + 1
        FREND = LNBR
        NODE = ORDER(FRBEG)
C
        IF (FRBEG .EQ. FREND) GO TO 500
        IF (FRBEG .GT. FREND   .OR.
     *      FREND .EQ. CCSIZE       ) RETURN
C
        MINFW = FDEG(NODE)
        NP = FRBEG
        DO 400 J= FRBEG, FREND
           JFDEG = FDEG(ORDER(J))
           IF (MINFW .LE. JFDEG) GO TO 400
C
           NP = J
           MINFW = JFDEG
400     CONTINUE
C
C       SHIFT NODES IN ORDER(FRBEG),...,ORDER(NP-1)
C       TO PRESERVE THE ORIGINAL ORDER OF THE FRONTAL NODES.
C
        NODE = ORDER(NP)
        CALL RCOPY( NP-FRBEG, ORDER(FRBEG), ORDER(FRBEG+1))
        ORDER(FRBEG) = NODE
C
C       FIND THE MASKED NBRS OF NODE AND PUT THEM INTO THE FRONT.
C
500     IF (FDEG(NODE) .LE. 0) GO TO 300
        CALL FNDNBR(NODE, ADJNCY, NR, DEG, MASK, ORDER, LNBR)
        GO TO 100
C
      END
```

```
C*****************************************************************
C
C       'SUBDEG' IS USED TO SUBTRACT ONE FROM THE FRONT
C       DEGREES (IN 'FDEG') OF ALL THE NEIGHBORS OF 'NODE'.
C
C*****************************************************************
C
      SUBROUTINE SUBDEG( NODE, ADJNCY, NR, DEG, FDEG)
        INTEGER ADJNCY(NR,1), DEG(1), FDEG(1)
C
        NDEG = DEG(NODE)
        IF (NDEG .LE. 0) RETURN
C
        DO 100 J= 1, NDEG
           NBR = ADJNCY(J,NODE)
           FDEG(NBR) = FDEG(NBR) - 1
100     CONTINUE
C
        RETURN
      END
```

```
C*****************************************************************
C
C     SUBROUTINE GENKNG FINDS THE KING ORDERING FOR A GENERAL
C     GRAPH.  FOR EACH CONNECTED COMPONENT, GENKNG FINDS THE
C     ORDERING USING THE ROUTINE 'KING'.
C
C*****************************************************************
C
      SUBROUTINE GENKNG(ADJNCY, NR, N, DEG, MASK,
     *                  LS, XLS, LASTLV, FDEG, PERM)
     *    INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *            LS(1), XLS(1), LASTLV(1), PERM(1), FDEG(1)
      INTEGER ROOT, CCSIZE
C
C     TURN 'MASK' OF ALL NODES TO 1.
C
      DO 100 I=1, N
         MASK(I) = 1
         FDEG(I) = DEG(I)
100   CONTINUE
C
      NUM = 1
C
C     DETERMINE A MASKED NODE 'ROOT' OF MIN DEGREE.
C
      DO 400 I=1, N
         IF (MASK(I) .NE. 1) GO TO 400
         ROOT = MINDEG(I, N, MASK, DEG)
C
C        FIRST FIND A PSEUDO PERIPHERIAL NODE.  'NSIZE' DENOTES
C        THE SIZE OF THE PRESENT CONNECTED COMPONENT.
C
         CALL FNROOT(ROOT, ADJNCY, NR, N, DEG, MASK,
     *               LS, XLS, NLVL, LASTLV)
         CCSIZE = XLS(NLVL+1) - 1
C
C        KING IS CALLED TO ORDER THE COMPONENT CONTAINING
C        THE NODE 'ROOT'.
C
         CALL KING(ROOT, ADJNCY, NR, DEG, MASK, CCSIZE, LS, FDEG)
C
         DO 300 J= 1,CCSIZE
            PERM(LS(J)) = NUM
            NUM = NUM + 1
300      CONTINUE
C
         IF (NUM .GT. N) RETURN
400   CONTINUE
      RETURN
      END
```

```
C*****************************************************************
C
C     SUBROUTINE 'LEVY' GENERATES THE LEVY ORDERING.
C
C*****************************************************************
C
      SUBROUTINE LEVY(ADJNCY, NR, N, DEG, MASK,
     *                ORDER, FDEG, PERM)
     *    INTEGER ADJNCY(NR,1), DEG(1), MASK(1),
     *            ORDER(1), FDEG(1), PERM(1)
      INTEGER NODE, NP, PTR
C
      DO 100 I= 1,N
         MASK(I) = 1
         ORDER(I) = I
         FDEG(I) = DEG(I)
100   CONTINUE
C
      PTR = 0
C
C     FROM THE SET OF UNLABELLED NODES (PTR,..,N),
C     SELECT A NODE WITH MINIMUM INCREASE IN FRONTWIDTH
C
200   PTR = PTR + 1
      IF (PTR .GE. N) GO TO 500
C
      MINFW = N
      DO 300 J=PTR, N
         JFDEG = FDEG(ORDER(J))
         IF (MINFW .LE. JFDEG) GO TO 300
         NP = J
         MINFW = JFDEG
         IF (MINFW .LE. JFDEG) GOTO 320
300   CONTINUE
C
C     SWITCH ORDER(NP) WITH ORDER(PTR).
C
320   NODE = ORDER(NP)
      ORDER(NP) = ORDER(PTR)
      ORDER(PTR) = NODE
C
C     IF THE LABELLED 'NODE' IS NOT IN THE FRONT,
C     UPDATE THE 'FDEG' OF ITS NEIGHBORS.
C
      IF (MASK(NODE) .EQ. 0) GO TO 350
      CALL SUBDEG(NODE, ADJNCY, NR, DEG, FDEG)
      MASK(NODE) = 0
C
C     FOR EACH NEW NODE IN THE FRONT, (MASKED NEIGHBOR
C     OF CURRENT 'NODE'), UPDATE THE 'FDEG' OF ITS NBRS.
C
350   NDEG = DEG(NODE)
      IF (NDEG .EQ. 0) GO TO 200
C
      DO 400 J= 1,NDEG
         NBR = ADJNCY(J, NODE)
         IF (MASK(NBR) .EQ. 0) GO TO 400
         CALL SUBDEG(NBR, ADJNCY, NR, DEG, FDEG)
         MASK(NBR) = 0
         FDEG(NBR) = FDEG(NBR) - 1
400   CONTINUE
C
      GO TO 200
C
500   DO 600 I=1,N
         PERM(ORDER(I)) = I
600   CONTINUE
C
      RETURN
      END
```

5 Element Annihilation Routines

This set of routines implements the EA algorithm as described in section 6.4.2. The following variables are used to represent a finite element graph.

NV ---

is the number of nodes in a triangular element. It is the row dimension of the elment structure TELMNT.

NT ---

is the number of triangular elements in the mesh.

TDUAL(3,NT) ---

is the connection table for the dual graph.

TDEG(NT) ---

is the degree of each element in the dual.

TELMNT(NV,NT) ---

is the element structure of the finite element graph. The nodes in triangle i are given by:

TELMNT(1,i), ... , TELMNT(NV,i).

** PARTAS(ROOT,TDUAL,TDEG,MASK,CCSIZE,ASEQ,ASPTR,EFRONT)

Purpose: This routine finds the annihilation sequence of a connected component in the dual graph. The algorithm minimizing local edge frontwidth is used.

Parameters:

ROOT --- is the starting element for the annihilation sequence. It also defines the connected component.

TDUAL, TDEG --- are the input dual graph parameters.

MASK(NT) --- On return, all the numbered elements have their MASK value changed to 0.

CCSIZE --- is an input integer containing the size of the connected component.

ASEQ(NT) --- is the vector for the annihilation sequence.

ASPTR --- is an integer pointer to the vector ASEQ. On input, ASPTR+1 is the location in ASEQ for the first element in the new annihilation sequence. On output, it points to the last element.

EFRONT(NT) --- is a working vector containing the current edge frontwidth of the triangular elements.

Subroutines used:

"RCOPY".

** FINDAS(TDUAL,NT,TDEG,MASK,LS,XLS,LASTLV,EFRONT,ASEQ)

Purpose: The subroutine is used to find an annihilation sequence of a given dual graph using minimum edge frontwidth approach.

Parameters:

TDUAL, NT, TDEG, MASK --- are the input dual graph parameters.

LS, XLS, LASTLV --- are the working vectors for level structures. They are used by "FNROOT" to find a starting element.

EFRONT(NT) --- is a working vector used by "PARTAS".

ASEQ(NT) --- is the output annihilation sequence. ASEQ(i) is the i-th element numbered in the sequence.

Subroutines used:

"MINDEG", "FNROOT", "PARTAS".

** EA(TDUAL,NT,TDEG,TELMNT,NV,N,MASK,LS,XLS,LASTLV,NTEMP,ASEQ,PERM)

Purpose: The subroutine numbers a given finite element graph by using an induced ordering of some annihilation sequence.

Parameters:

TDUAL, NT, TDEG, TELMNT, NV, N, MASK --- are the input finite element graph parameters.

LS, XLS, LASTLV --- are the working vectors for level structures.

NTEMP(*) --- is a working vector of length at least maximum(NT,N).

ASEQ(NT) --- is the vector to store the annihilation sequence.

PERM(N) --- is the permutation vector, where PERM(i) is the new label assigned to node i in the new labelling.

Subroutines used:

"FINDAS".

```
C*****************************************************************
C
C     PARTAS FINDS PART OF AN ANNIHILATION SEQUENCE.
C     THE PART FOUND CORRESPONDS TO A CONNECTED COMPONENT OF
C     THE DUAL GRAPH.
C
C*****************************************************************
C
      SUBROUTINE PARTAS( ROOT, TDUAL, TDEG, MASK, CCSIZE,
     *                   ASEQ, ASPTR, EFRONT)
     *    INTEGER TDUAL(3,1), TDEG(1), MASK(1),
     *            ASEQ(1), EFRONT(1)
     *    INTEGER FRBEG, FREND, ASPTR, ROOT, CCSIZE,
     *            MINEFW, FEFR, TP, ELEMNT, CTDEG, TNBR
C
C     'FRBEG' AND 'FREND' POINTS TO THE BEGINNING AND
C     THE END OF THE CURRENT FRONT.
C
      FRBEG = 0
      ASPTR = ASPTR + 1
C
      ASEQ(ASPTR) = ROOT
      MASK(ROOT) = 0
C
100   FRBEG = FRBEG + 1
      FREND = ASPTR
      ELEMNT = ASEQ(FRBEG)
C
      IF (FRBEG .EQ. FREND) GO TO 400
      IF (FRBEG .GT. FREND  .OR.
     *    FREND .EQ. CCSIZE)  RETURN
C
C     FROM THE CURRENT FRONT, SELECT THE TRIANGLE WHICH
C     INCREASES THE EDGE FRONT LEAST.
C
      MINEFW = 4
      DO 200 J= FRBEG,FREND
         JEFR = EFRONT(ASEQ(J))
         IF (MINEFW .LE. JEFR) GO TO 200
         TP = J
         MINEFW = JEFR
200   CONTINUE
C
      ELEMNT = ASEQ(TP)
      CALL RCOPY(TP-FRBEG, ASEQ(FRBEG), ASEQ(FRBEG+1))
      ASEQ(FRBEG) = ELEMNT
C
C     UPDATE 'EFRONT' VALUE OF THE NBRS OF 'ELEMNT' AND
C     PUT THE MASKED NBRS OF 'ELEMNT' INTO THE FRONT.
C
C
400   CTDEG = TDEG(ELEMNT)
      IF (CTDEG .EQ. 0) GO TO 100
C
      DO 500 J= 1,CTDEG
         TNBR = TDUAL(J,ELEMNT)
         EFRONT(TNBR) = EFRONT(TNBR) - 2
C
         IF (MASK(TNBR) .EQ. 0) GO TO 500
         ASPTR = ASPTR + 1
         ASEQ(ASPTR) = TNBR
         MASK(TNBR) = 0
500   CONTINUE
      GO TO 100
C
      END
```

```
C*****************************************************************
C
C     FINDAS FINDS AN ANNIHILATION SEQUENCE FOR THE GIVEN
C     DUAL GRAPH. THE ALGORITHM THAT MINIMIZES LOCAL EDGE
C     FRONTWIDTH IS USED.
C
C*****************************************************************
C
      SUBROUTINE FINDAS( TDUAL, NT, TDEG, MASK,
     *                   LS, XLS, LASTLV, EFRONT, ASEQ)
     *    INTEGER TDUAL(3,1), TDEG(1), MASK(1),
     *            LS(1), XLS(1), LASTLV(1), EFRONT(1), ASEQ(1)
     *    INTEGER ASPTR, ROOT, CCSIZE
C
C     INITIALIZE 'MASK' AND 'EFRONT'.  EFRONT(I) STORES
C     THE CURRENT INCREASE IN EDGE FRONT FOR ELEMENT I.
C
      DO 100 I=1,NT
         MASK(I) = 1
         EFRONT(I) = TDEG(I)
100   CONTINUE
C
      ASPTR = 0
      DO 200 I= 1,NT
         IF (MASK(I) .NE. 1) GO TO 200
         ROOT = MINDEG(I, NT, MASK, TDEG)
C
         CALL FNROOT( ROOT, TDUAL, 3, NT, TDEG, MASK,
     *                LS, XLS, NLVL, LASTLV)
         CCSIZE = XLS(NLVL+1) - 1
C
         CALL PARTAS( ROOT, TDUAL, TDEG, MASK, CCSIZE,
     *                ASEQ, ASPTR, EFRONT)
C
         IF (ASPTR .GE. NT) RETURN
200   CONTINUE
C
      RETURN
      END
```

```
C*****************************************************************
C
C     EA NUMBERS THE GIVEN FINITE ELEMENT GRAPH BY FIRST
C     DETERMINING AN ANNIHILATION SEQUENCE USING 'FINDAS',
C     AND THEN PRODUCES AN INDUCED ORDERING ON THE NODES.
C
C*****************************************************************
C
      SUBROUTINE EA(TDUAL, NT, TDEG, TELMNT, NV, N, MASK,
     *              LS, XLS, LASTLV, NTEMP, ASEQ, PERM)
     *    INTEGER TDUAL(3,1), TDEG(1), TELMNT(NV,1), MASK(1),
     *            LS(1), XLS(1), LASTLV(1),
     *            NTEMP(1), ASEQ(1), PERM(1)
     *    INTEGER ELEMNT, NUM
C
C     AN ANNIHILATION SEQ 'ASEQ' IN DETERMINED BY 'FINDAS'.
C
      CALL FINDAS(TDUAL, NT, TDEG, MASK,
     *            LS, XLS, LASTLV, NTEMP, ASEQ)
C
C     INITIALIZE 'NTEMP', WHERE NTEMP(NODE) IS NOW THE
C     NUMBER OF TRIANGLES TO WHICH 'NODE' BELONGS.
C
      DO 100 NODE= 1,N
         NTEMP(NODE) = 0
100   CONTINUE
C
      DO 200 I= 1,NT
         DO 200 J= 1,NV
            NODE = TELMNT(J,I)
            NTEMP(NODE) = NTEMP(NODE) + 1
200   CONTINUE
C
C     AN INDUCED NODE ORDERING 'PERM' IS OBTAINED BY
C     RUNNING THROUGH THE TRIANGULAR ELEMENTS IN THE
C     ORDER GIVEN BY 'ASEQ'.
C
      NUM = 1
      DO 400 I= 1,NT
         ELEMNT = ASEQ(I)
C
         DO 300 J= 1,NV
            NODE = TELMNT(J, ELEMNT)
            NTEMP(NODE) = NTEMP(NODE) - 1
C
            IF (NTEMP(NODE) .GT. 0) GO TO 300
            PERM(NODE) = NUM
            NUM = NUM + 1
300      CONTINUE
400   CONTINUE
C
      RETURN
      END
```