AN AUTOMATIC NESTED DISSECTION ALGORITHM FOR IRREGULAR FINITE ELEMENT PROBLEMS*

by

Alan George[†]

and

Joseph W. H. Liu[††]

Research Report CS-76-38

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

October 1976

[†] Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

[††] Computer Services Division
City of Mississauga
Mississauga, Ontario, Canada

* Research supported in part by Canadian National Research Council grant A8111.

ABSTRACT

A formal definition of a nested dissection ordering of the graph of a general sparse symmetric matrix A is given. After introducing some preliminary results which provide a direct relationship between the structures of A and its triangular factor L, where $A = LL^T$, some results about fill for nested dissection orderings are provided. Finally, a heuristic algorithm is described for finding a nested dissection ordering for an undirected graph, along with appropriate data structures and a storage allocation scheme for a linear equation solver to use such orderings. The ordering algorithm/linear equation solver combination is applied to the graphs of matrices arising in typical finite element applications, and numerical experiments are provided which indicate that our combination of ordering and solution schemes is superior to standard band or envelope schemes as long as the problems are moderately large.

§1 Introduction

In this paper we consider the problem of directly solving the system of linear equations

$$Ax = b, \tag{1.1}$$

where A is a sparse N by N positive definite matrix arising in the application of finite element methods. We characterize the structure of A more precisely below. The system (1.1) is solved using Cholesky's method by first factoring A into the product LL^T, where L is lower triangular, and then solving the triangular systems $Ly = b$ and $L^Tx = y$.

When a sparse matrix is factored, it normally suffers fill. Under the usual assumption that exact numerical cancellation does not occur, $L + L^T$ is usually fuller than A. For any N by N permutation matrix P, the matrix PAP^T is also positive definite and Cholesky's method is still applicable, so we could instead solve the equivalent system

$$(PAP^T)(Px) = Pb. \tag{1.2}$$

A judicious choice of P can often drastically reduce fill, so if zeros are exploited, this can imply a reduction in storage requirements and/or arithmetic requirements for the linear equation solver. The objective of this paper is to describe a heuristic algorithm for finding such a permutation P.

In [9], the first author described a nested dissection ordering for an n by n grid graph which reduced the arithmetic operation count for directly solving the associated n^2 by n^2 matrix problem from the usual

$O(n^4)$ to only $O(n^3)$. The corresponding fill was reduced from $O(n^3)$ to $O(n^2 \log_2 n)$. Since then several papers by various authors have appeared, dealing with various aspects of the ordering for square and rectangular problems [6,18]. In [10], it was demonstrated that these orderings could be implemented in a highly efficient manner, with quite low storage and computational overhead.

However, little progress seems to have been made in finding automatic schemes for producing such efficient orderings for less regular problems, although intuitively it is clear what the general nature of the ordering should be. In this paper we describe a heuristic algorithm for producing nested dissection orderings for sparse matrix problems, and provide numerical experiments which demonstrate its effectiveness when applied to a class of two dimensional finite element problems, which we now characterize. A preliminary version of this work appeared in [12].

Let M be a planar mesh consisting of the union of triangles and/or quadrilaterals called elements, with adjacent elements having a common side or a common vertex. There is a node at each vertex of M, and there may also be nodes lying on element sides and faces, as shown in Figure 1.1.

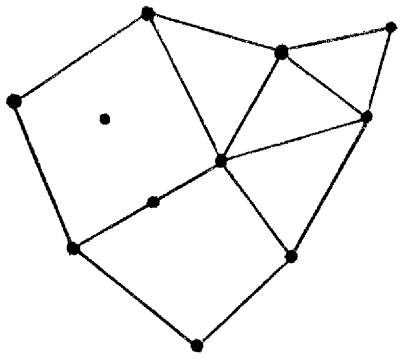


Figure 1.1 An 11 node finite element mesh with 6 elements.

Associated with each node is one or more variables x_i, and for some labelling of these N variables we define a finite element system $Ax = b$ associated with M as one for which A is symmetric and positive definite, and for which $A_{ij} \neq 0 \Rightarrow x_i$ and x_j are associated with nodes of the same element. The matrix problems thus generated correspond in type and structure to a large class of problems arising in various scientific and engineering applications [19].

An outline of the paper is as follows. In section 2 we review some graph theory results which pertain to symmetric Gaussian elimination, and provide some notations and results which are needed in section 3. Section 3 contains a formal definition of nested dissection orderings, and some results are proved about these orderings which are helpful in motivating the ordering algorithm. Section 4 contains a description of the storage scheme used in the linear equation solver which uses the ordering, along with some implementation details. Section 5 contains a description of the ordering algorithm, along with details on its computer implementation. Section 6 contains a description of our test problems and numerical experiments, and section 7 contains our concluding remarks.

§2 Preliminaries

2.1 Some Graph Theory Terminology

We begin by reviewing some basic concepts of graph theory which we need in this and the next section. An undirected graph $G = (X,E)$ consists of a finite non-empty set X of nodes or vertices together with a set E of edges, which are unordered pairs of distinct nodes of X. A graph $G' = (X',E')$ is a subgraph of G if $X' \subseteq X$ and $E' \subseteq E$. For $Y \subset X$, the section graph $G(Y)$ is the subgraph $(Y,E(Y))$, where

$$E(Y) = \{\{x,y\} \in E \mid x \in Y, y \in Y\}.$$

Two nodes x and y of G are adjacent if $\{x,y\} \in E$. For $Y \subset X$, the adjacent set of Y, denoted by $Adj(Y)$, is:

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

When Y is a single node y, we write $Adj(y)$ rather than $Adj(\{y\})$. The degree of the node x in G is the number $|Adj(x)|$, where $|S|$ is the cardinality of the finite set S.

A graph is complete if every pair of vertices is adjacent. A clique in G is a complete subgraph of G.

For distinct nodes x and y in G, a path from x to y of length k is an ordered set of distinct nodes $(v_1,v_2,\ldots,v_{k+1})$, where $x = v_1$ and $y = v_{k+1}$, such that $v_i \in Adj(v_{i+1})$, $i = 1,2,\ldots,k$. Note that k may be 1 in which case x and y are adjacent. A graph G is connected if for every pair of distinct vertices $x, y \in X$, there is at least one path from x to y. If G is disconnected, it consists of one or more connected components.

The set $Y \subset X$ is a separator of the connected graph G if $G(X\backslash Y)$ consists of two or more components; Y is a minimal separator is no proper subset of Y is a separator of G. The following lemma is immediate.

Lemma 2.1 Let Y be a minimal separator of G, and suppose its removal yields m components $G(X_i) = (X_i, E(X_i))$ for $1 \le i \le m$. Then for every $y \in Y$, $Adj(y) \cap X_i \neq \phi$. □

For our purpose in the study of Gaussian elimination, it is helpful to generalize the concept of adjacent set. Let $S \subset X$ and $y \notin S$. The node y is said to be reachable from a node x through S if there exists a path $(x, v_1, v_2, \ldots, v_k, y)$ from x to y such that $v_i \in S$ for $1 \le i \le k$. The reachable set of x through S, denoted by Reach (x,S), is then defined to be

$$\text{Reach } (x,S) = \{y \in X \backslash S \mid y \text{ is reachable from } x \text{ through } S\}.$$

Note that paths may be only of length one, and S may be empty. When $S = \phi$, we have $\text{Reach } (x,\phi) = Adj(x)$, so that the reachable set of x through S may be regarded as a generalization of the adjacent set of x.

Two notions pertinent to the study of nested dissections are partitionings and orderings. They are now defined and some related terms are introduced.

A partitioning P of the graph G is a subset of the power set of X:

$$P = \{Y_1, Y_2, \ldots, Y_k\},$$

satisfying $\bigcup_{i=1}^{k} Y_i = X$ and $Y_i \cap Y_j = \phi$ for $i \neq j$.

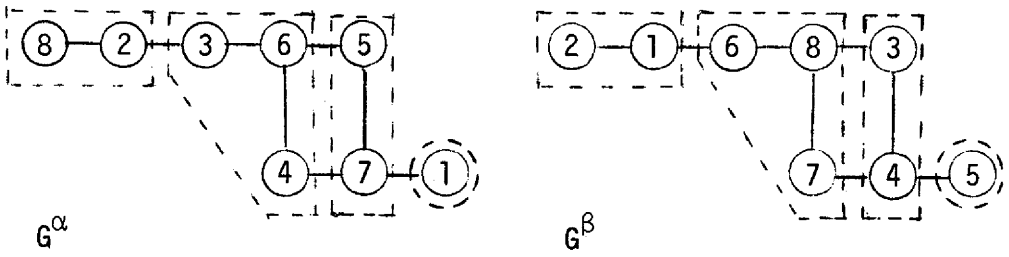
Let G = (X,E) be a graph with $|X| = N$. An ordering (numbering, labelling) of G is a bijective mapping $\alpha: \{1,2,\ldots,N\} \to X$.

An ordering α is said to be compatible with the partitioning P if for each $Y \in P$, α numbers nodes in Y consecutively.

Consider two node orderings α and β on the graph. The ordering α is said to conform with β on P if for $x,y \in Y \in P$, then

$$\alpha(x) < \alpha(y) \iff \beta(x) < \beta(y).$$

In other words, for each partition member in $\mathcal{P}$, its nodes appear in the same relative order in both orderings. Figure 2.1 illustrates two orderings that conform with each other on the partitioning as shown. Note that β is compatible with the partitioning.



G^{α} G^{β}

Figure 2.1 Two conforming orderings on a partition.

In a disconnected graph, the connected components define a natural partitioning on the graph. For our subsequent discussion, we introduce the following definition. Let Y be a subset of the node set X. The component partitioning $C(Y)$ of Y is defined as:

$C(Y) = \{S \subset Y |$ **G(S) is a connected component in the section graph G(Y)**$\}$.

Thus, a graph $G = (X,E)$ **is connected if and only if** $C(X) = \{X\}$; **a subset** S is a separator if and only if $|C(X \backslash S)| > 1$.

Another important type of graph partitioning is the class of level structures [2]. A level structure of a connected graph $G = (X,E)$ is a partitioning

$$\mathcal{L} = \{L_0, L_1, \ldots, L_\ell\}$$

of the node set X such that $Adj(L_0) \subset L_1$, $Adj(L_\ell) \subset L_{\ell-1}$, and $Adj(L_i) \subset L_{i-1} \cup L_{i+1}$ for $0 < i < \ell$. Note that for $0 < i < \ell$, the set L_i is a separator of G.

In particular, the notion of a rooted level structure is an important construct in the algorithm presented in section 5. For $x \in X$, the rooted level structure at x is the level structure

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

where

$$L_0(x) = \{x\}$$
$$L_i(x) = \mathrm{Adj}\Big(\bigcup_{k=0}^{i-1} L_k(x)\Big).$$

The quantity $\ell(x)$ is sometimes called the eccentricity of the node x. The diameter of G is then defined as

$$\delta(G) = \max \{\ell(x) \ : \ x \in X\}.$$

A peripheral node x is one such that $\ell(x) = \delta(G)$.

2.2 Graphs of Matrices and Symmetric Gaussian Elimination

Let A be an N by N symmetric matrix. The labelled undirected graph of A, denoted by $G^A = (X^A, E^A)$, is one for which X^A is labelled from 1 to N and $\{x_i, x_j\} \in E^A$ if and only if $A_{ij} \neq 0$, $i > j$. The unlabelled graph of A is simply G^A with its labels removed. (Here we use the notation x_i to denote $\alpha(i)$, where α is the labelling of the graph implied by A). For any N by N permutation matrix P, the unlabelled graphs of A and PAP^T are the same, but the associated labellings differ. Thus, finding a good ordering for A can be viewed as finding a good labelling for G^A.

Consider the symmetric factorization of the matrix A into LL^T. For $i = 1,2,\ldots, N$ let

$$\nu_i = |\{L_{ji} : L_{ji} \neq 0, \ j > i\}|.$$

We quote the following result from Rose [17].

Lemma 2.2 The number of off-diagonal nonzeros in L is given by

$$\sum_{i=1}^{N} \nu_i ,$$

and the number of multiplicative operations required for the factoriation is

$$\tfrac{1}{2} \sum_{i=1}^{N} \nu_i(\nu_i + 3).$$

□

In previous graph theory studies of symmetric Gaussian elimination, the process has been viewed as a sequence of elimination graphs $G = G_0, G_1, G_2, \ldots G_{N-1}$ whose node sets decrease by one at each step and the structure of G_i reflects the structure of the matrix remaining to be factored after the i-th step of the factorization. (The interested reader is referred to [17] for details.) The number ν_i is simply the degree of node x_i in G_{i-1}.

This approach to the analysis has the disadvantage that the quantities of interest, namely the ν_i, are in a sense only obtained "a-posteriori"; the number ν_i is only known after step i-1 of the factorization has been completed (or at least simulated). A much more desirable situation is to have some direct relations between the numbers ν_i and the original matrix A, and this is the reason for the notions and results we now introduce.

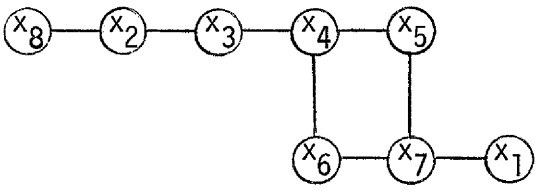
Let $G^A = (X^A, E^A)$ and $G^F = (X^F, E^F)$ where $F = L + L^T$. Here G^F is called the filled graph of G^A, and E^F consists of all the edges E^A in G^A together with all the edges added during the factorization. Obviously $X^F = X^A$, and the edge sets E^A and E^F are related by the following lemma due to Parter [16].

Lemma 2.3 The unordered pair $\{x_i,x_j\} \in E^F$ if and only if $\{x_i,x_j\} \in E^A$ or $\{x_i,x_k\} \in E^F$ and $\{x_j,x_k\} \in E^F$ for some $k < \min\{i,j\}$. □

Note that this lemma does not really meet the objection raised above since its statement is "recursive" in E^F. Our objective in what follows is to develop a characterization of the edge set E^F which depends only on G^A. In this direction, we introduce the notion of the elimination adjacent set of x_i, defined by

$$Eadj(x_i) = Reach\ (x_i,\{x_1,x_2,\ldots,x_{i-1}\}).$$

In other words, $x_j \in Eadj(x_i)$ if and only if $j > i$ and there is a path joining x_i to x_j in the section graph $G(\{x_1,x_2,\ldots,x_i\} \cup \{x_j\})$. Figure 2.2 illustrates the definition of $Eadj(x_i)$.



$Eadj\ (x_1) = \{x_7\}$

$Eadj\ (x_2) = \{x_3,x_8\}$

$Eadj\ (x_3) = \{x_4,x_8\}$

$Eadj\ (x_4) = \{x_5,x_6,x_8\}$

$Eadj\ (x_5) = \{x_6,x_7,x_8\}$

Figure 2.2 Example to illustrate Eadj.

Note that the notion of reachable set introduced in section 2.1 is defined for unordered graphs, but the notion of elimination adjacent set is defined only for an ordered graph. (Recall that in the above, $x_i \equiv \alpha(i)$) One of our objectives is to study the elimination adjacent sets for different orderings, say α and β, for a given graph G. When it is not clear from context what ordering is intended, we will write $Eadj(x,\alpha)$ or $Eadj(\alpha(i))$.

Lemma 2.4 Let α be an ordering on the graph $G = (X,E)$ and $\alpha(i) = x_i$. Then $Eadj(x_i) = Adj(S)$, where $S \in C(\{x_1,\ldots,x_i\})$ and $x_i \in S$.

Proof: For $y \in Eadj(x_i)$, we have a path $(x_i,y_1,\ldots,y_t,y)$ where $y_k \in \{x_1,\ldots,x_{i-1}\}$, $1 \leq k \leq t$. By the definition of the set S, $y_t \in S$ so that $y \in Adj(S)$.

On the other hand, consider $y \in Adj(S)$. If $y \in Adj(x_i)$, clearly $y \in Eadj(x_i)$. For $y \in Adj(s)$, where $s \in S$, a path in S can be found from x_i to s and hence to y. Therefore $y \in Eadj(x_i)$. □

Lemma 2.5 Let $j > i$. The unordered pair $\{x_i,x_j\} \in E^F$ if and only if $x_j \in Eadj(x_i)$.

Proof: "if part": Assume $x_j \in Eadj(x_i)$. If $x_j \in Adj(x_i)$ in G there is nothing to prove since $E^A \subseteq E^F$. If $x_j \notin Adj(x_i)$, by assumption there exists a path $\{x_i,y_1,y_2,\ldots,y_t,x_j\}$ in G with $y_k \in \{x_1,x_2,\ldots,x_{i-1}\}$ and $t \geq 1$. If $t = 1$, the result follows immediately from lemma 2.3. If $t > 1$ a simple induction on t, using lemma 2.3, shows that $\{x_i,x_j\} \in E^F$.

"only if part": When $i = 1$, $\{x_i,x_j\} \in E^F$ implies that $\{x_i,x_j\} \in E^A$ by lemma 2.3, so that $x_j \in Eadj(x_1)$. Suppose the result holds for subscripts less than i, and that $\{x_i,x_j\} \in E^F$. If $\{x_i,x_j\} \in E^A$, then clearly $x_j \in Eadj(x_i)$. Otherwise, by lemma 2.3, $\{x_i,x_j\} \in E^F$ implies there exists a $k < \min\{i,j\}$ such that $\{x_i,x_k\} \in E^F$ and $\{x_k,x_j\} \in E^F$. By the inductive assumption $x_i \in Eadj(x_k)$ and $x_j \in Eadj(x_k)$, so that a path can be found from x_i to x_j in the subgraph $G(\{x_1,x_2,\ldots,x_k\} \cup \{x_i,x_j\})$. That is, $x_j \in Reach(x_i,\{x_1,x_2,\ldots,x_k\})$, which implies that $x_j \in Reach(x_i,\{x_1,x_2,\ldots,x_{i-1}\}) \equiv Eadj(x_i)$. □

Corollary 2.6 For $i = 1,2,\ldots,N$, $\nu_i = |Eadj(x_i)|$. □

Corollary 2.7 The section graph $G^A(Eadj(x_i) \cup \{x_i\})$ is a clique in the filled graph G^F.

Proof: For $y \in Eadj(x_i)$, it follows from lemma 2.5 that $\{x_i,y\} \in E^F$.

Consider x_j, $x_k \in Eadj(x_i)$ where $k > j$. From definition, we have two paths $(x_i,y_1,\ldots,y_r,x_j)$ and $(x_i,z_1,\ldots,z_s,x_k)$, where y's and z's belong to $\{x_1,\ldots,x_{i-1}\}$. On combining these two paths, we get $(x_j,y_r,\ldots,y_1,x_i,z_1,\ldots,z_s,x_k)$ which goes from x_j to x_k. By definition, $x_k \in Eadj(x_j)$. Again, lemma 2.5 implies $\{x_j,x_k\} \in E^F$. □

To facilitate our discussion, we introduce these two notations for an ordered graph:

$$\eta(G,\alpha) = \sum_{i=1}^{N} |Eadj(\alpha(i))|$$

$$\theta(G,\alpha) = \tfrac{1}{2} \sum_{i=1}^{N} |Eadj(\alpha(i))| \{|Eadj(\alpha(i))| + 1\}.$$

They have the following important interpretations: when G is associated with some matrix A, $\eta(G,\alpha)$ corresponds to the number of off-diagonal non-zeros in the triangular factor of A and $\theta(G,\alpha)$ is the number of multiplicative operations required to perform the symmetric factorization of A (see lemma 2.2 and corollary 2.6). Two orderings α and β of a graph G are said to be equivalent if $\theta(G,\alpha) = \theta(G,\beta)$ and $\eta(G,\alpha) = \eta(G,\beta)$.

The following lemma is useful in establishing the equivalence of orderings.

Lemma 2.8 Let α and β be two orderings on the graph $G = (X,E)$. Consider $x \in X$ where $\alpha(i) = x = \beta(j)$. Then $Eadj(x,\alpha) = Eadj(x,\beta)$ if and only if there exists a subset $S \subset X$ with $x \in S$ such that

$$S \cap \{\alpha(1),\ldots,\alpha(i)\} = S \cap \{\beta(1),\ldots,\beta(j)\},$$

and $$Adj(S) \cap \{\alpha(1),\ldots,\alpha(i)\} = Adj(S) \cap \{\beta(1),\ldots,\beta(j)\} = \phi.$$

Proof: "if" part: Assume that such a subset S exists. Consider any $y \in Eadj(x,\alpha)$. There is a path $(x,y_1,\ldots,y_t,y)$ where $y_k \in \{\alpha(1),\ldots,\alpha(i-1)\}$. The nodes $y_k \in S$ for otherwise $Adj(S) \cap \{\alpha(1),\ldots,\alpha(i)\} \neq \phi$. Therefore $y_k \in S \cap \{\alpha(1),\ldots,\alpha(i)\} = S \cap \{\beta(1),\ldots,\beta(j)\}$. It remains to show that $y \notin \{\beta(1),\ldots,\beta(j)\}$. Assume for contradiction that $y \in \{\beta(1),\ldots,\beta(j)\}$. The node $y \notin S$, for otherwise

$$y \in S \cap \{\beta(1),\ldots,\beta(j)\} = S \cap \{\alpha(1),\ldots,\alpha(i)\},$$

implying that $y \notin Eadj(x,\alpha)$. But then $y \in Adj(y_t)$ so that $y \in Adj(S) \cap \{\beta(1),\ldots,\beta(j)\}$, which is again a contradiction.

Therefore $y \in Eadj(x,\beta)$; that is, $Eadj(x,\alpha) \subseteq Eadj(x,\beta)$. By symmetry, we have the other inclusion.

"only if" part: Assume $Eadj(x,\alpha) = Eadj(x,\beta)$. We take $S \in C(\{\alpha(1),\ldots,\alpha(i)\})$ where $x \in S$. The reader is left to verify that S satisfies the properties in the lemma. □

This lemma is slightly more general than we need in the sequel. When we use it in section 3, the set S will always be as defined in the "only if" part of the proof, that is, $S \in C(\{\alpha(1),\ldots,\alpha(i)\})$ and $\alpha(i) \in S$.

2.3 Finding Pseudo-Peripheral Nodes of a Graph

The previous section associates symmetric matrices with undirected graphs so that to find a good permutation for a matrix is equivalent to obtaining a good ordering on the corresponding graph.

The effectiveness of many ordering algorithms, including the one described in this paper, depends quite critically on the proper choice of a "starting node". Experience has led researchers to advocate the use of peripheral nodes to start some algorithms. Unfortunately, no efficient algorithm is known to determine such nodes of a general graph.

In [13], Gibbs, Poole and Stockmeyer devised an algorithm which appears to be extremely effective in finding nodes whose eccentricity $\ell(x)$ is close to the diameter of the graph. Such nodes will be called pseudo-peripheral nodes. The algorithm is described below.

Step 1: Find a node r of minimum degree.

Step 2: Generate the rooted level structure at r:

$$\mathcal{L}(r) = \{L_0(r), L_1(r), \ldots, L_{\ell(r)}(r)\}.$$

Step 3: Sort $L_{\ell(r)}(r)$ in increasing order of degree.

Step 4: For each $x \in L_{\ell(r)}(r)$ in order of increasing degree, generate the rooted level structure $\mathcal{L}(x)$. If $\ell(x) > \ell(r)$, put $r \leftarrow x$ and go to step 3.

Step 5: r is a pseudo-peripheral node.

In their actual implementation [4], Gibbs et al introduced a "short circuit" technique, where wide level structures are rejected as soon as they are detected. In other words, for many of the $x \in L_{\ell(r)}(r)$ in step 4, only part of the level structure $\mathcal{L}(x)$ will be generated.

However, for some problems, the set $L_{\ell(r)}(r)$ will be quite large, so that many level structures or partial level structures will be generated in step 4. Therefore, in our implementation, instead of the above short-circuit technique, we modify the algorithm by "shrinking" the connected components of $G(L_{\ell(r)}(r))$ to a single node of minimum degree. Steps 3 and 4 are replaced by the following.

Step 3': Find all the connected components in $L_{\ell(r)}(r)$, that is, $C(L_{\ell(r)}(r))$.

Step 4': For each $C \in C(L_{\ell(r)}(r)$, find a node x of minimum degree in C and generate its rooted level structure $\mathcal{L}(x)$. If $\ell(x) > \ell(r)$, put $r \leftarrow x$ and go to step 3'.

In addition to the above changes, we also found that an equally good Step 1 was simply to choose an arbitrary node r, rather than one of minimum degree, so that is what our algorithm does. For our applications, this modification improved the execution time substantially and did not affect the outcome much. The readers are referred to [11] for a more detailed study of these modifications. In the sequel, we will refer to nodes found by this modified algorithm as pseudo-peripheral nodes.

§3 Graph Theoretic Formulation of Nested Dissection

3.1 Nested Dissection Partitionings and Orderings

In this section, we formally define what we mean by a nested dissection ordering, in preparation for a description of an algorithm for producing such orderings. Some related results are also established.

A nested dissection ordering of a graph $G = (X,E)$ is defined as follows. Let $R_0 = X$, and for $m = 0,1,\ldots,h$ until $R_{h+1} = \phi$ do the following:

a) Determine the connected components of R_m and label them $R_m^1, R_m^2, \ldots, R_m^{r_m}$.

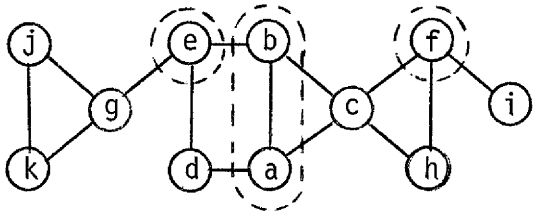
b) For $j = 1,\ldots,r_m$, choose $S_m^j \subseteq R_m^j$ such that S_m^j is either a separator of $G(R_m^j)$ or is equal to R_m^j. Define $S_m = \bigcup_{j=1}^{r_m} S_m^j$.

c) Define $R_{m+1} = R_m \setminus S_m$.

The partitioning $P = \{S_m^j \subset X: 1 \leq j \leq r_m \text{ and } 0 \leq m \leq h\}$ of X thus defined is called a nested dissection partitioning (ND-partitioning). The quantity h, defined by

$$h = \max \{m: R_m \neq \phi\},$$

is referred to as the dissection height.



$G = (X,E)$

$R_0 = X$ $\qquad S_0 = \{a,b\}$

$R_1^1 = \{c,f,h,i\}$ $\qquad S_1^1 = \{f\}$

$R_1^2 = \{d,e,g,j,k\}$ $\qquad S_1^2 = \{e\}$

$R_2^1 = \{d\}$

$R_2^2 = \{g,j,k\}$ $\qquad S_2^j = R_2^j \quad j = 1,2,3,4.$

$R_2^3 = \{i\}$

$R_2^4 = \{c,h\}$

Figure 3.1 A nested dissection partitioning $\{S_m^j\}$.

In the study of nested dissection orderings, it is instructive to associate a dissection tree with the partitioning P. We first make some preliminary definitions.

A tree is a graph where every pair of distinct nodes is connected by a unique path. For a tree $T = (X,E)$, $|X| = |E| + 1$. A rooted tree is a tree T with a distinguished node r, called the root of T. If the path from the root r to the node y passes through the node x, x is said to be an ancestor of y and y a descendant of x.

If, in addition, $x \in \text{Adj}(y)$, x is the father of y and y a son of x. The section graph of a node x and its descendants is called the subtree at x.

Let $P = \{S_m^j\}$ be a nested dissection partitioning. The dissection tree associated with P is the tree (P,T) rooted at S_0 where S_m^j is the father of S_{m+1}^k if and only if $R_m^j \supset R_{m+1}^k$. Note that we are using the one-to-one correspondence between the sets $\{S_m^j\}$ and $\{R_m^j\}$.

Figure 3.2 contains an example of a dissection tree. It should be noted that the subtree rooted at S_m^j in the dissection tree corresponds to the vertex set R_m^j. For example, R_1^1 is given by

$$S_1^1 \cup S_2^1 \cup S_2^2,$$

and $\{S_1^1, S_2^1, S_2^2\}$ forms the subtree at S_1^1.

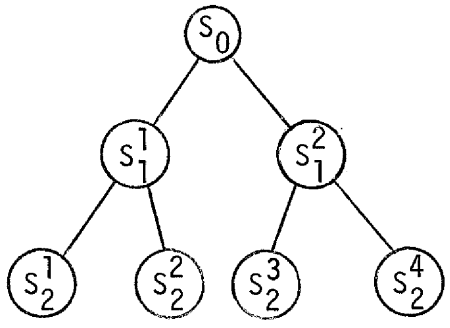


Figure 3.2 Dissection tree for the nested dissection partitioning in Figure 3.1.

In figure 3.2, those S_m^j having no descendant satisfy $S_m^j = R_m^j$. In the sequel, they will be referred to as terminal members of P. Those S_m^j which are separators of $G(R_m^j)$ will sometimes be referred to as non-terminal members.

A ND-partitioning $P = \{S_m^j\}$ is said to be complete if $G(S_m^j)$ is a clique in the graph for every terminal member S_m^j of P. That is, in step b of the operational definition above, a separator is chosen whenever possible. From the definition of S_m^j, we have the following lemma.

Lemma 3.1 Let S_{m-1}^k be the separator chosen in $G(R_{m-1}^k)$ and $R_m^j \in C(R_{m-1}^k \setminus S_{m-1}^k)$. Then $Adj(R_m^j) \subset Adj(R_{m-1}^k) \cup S_{m-1}^k$. □

An ordering α of X is said to be a nested dissection ordering with respect to $P = \{S_m^j\}$ if for $x \in S_m^j$ and $y \in R_m^j \setminus S_m^j$, $\alpha^{-1}(x) > \alpha^{-1}(y)$. An example of a nested dissection ordering on the partitioning of figure 3.1 is shown in figure 3.3.

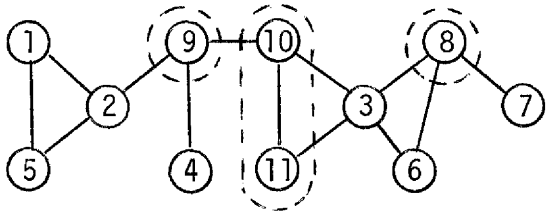


Figure 3.3 A nested dissection ordering on the nested dissection partitioning of figure 3.1

Note that an ordering that numbers the nodes of each partition member S_m^j before the nodes of its father in the dissection tree is a nested dissection ordering.

Lemma 3.2 Let α be a nested dissection ordering for $P = \{S_m^j\}$, a ND-partitioning. Then α numbers nodes in R_m^j before those in $Adj(R_m^j)$.

Proof: We prove the lemma by induction on m. The result clearly holds when m = 0. Assume that it is true for all $k < m$. Let $R_m^j \in G(R_{m-1}^k \backslash S_{m-1}^k)$. By the induction assumption, α numbers R_{m-1}^k before $Adj(R_{m-1}^k)$; and by definition, α numbers $R_{m-1}^k \backslash S_{m-1}^k$ before S_{m-1}^k. But $R_m^j \subset R_{m-1}^k \backslash S_{m-1}^k$, so that α numbers R_m^j before $Adj(R_{m-1}^k) \cup S_{m-1}^k$. The result then follows from lemma 3.1. □

For a given nesting dissection ordering α on some $P = \{S_m^j\}$, it is interesting to consider the corresponding elimination adjacent sets as introduced in section 2.

Lemma 3.3 Let $x \in R_m^j$. Then

$$Eadj(x,\alpha) \subset Adj(R_m^j) \cup R_m^j.$$ □

Theorem 3.4 Let $x \in S_m^j$. Then

$$Eadj(x,\alpha) \subset Adj(R_m^j) \cup S_m^j$$

Proof: Let $\alpha(i) = x$. By lemma 3.3, it remains to show $Eadj(x,\alpha) \cap (R_m^j \backslash S_m^j) = \phi$. Assume for contradiction that the intersection is nonempty, say $y \in Eadj(x,\alpha) \cap (R_m^j \backslash S_m^j)$. Let y belong to R_{m+1}^k, a component in $G(R_m^j \backslash S_m^j)$. But this implies the existence of an edge $\{p,q\}$ with $p \in Adj(R_{m+1}^k)$ and $q \in R_{m+1}^k$ and $\alpha^{-1}(p) < \alpha^{-1}(q)$ contradicting lemma 3.2. □

Theorem 3.5 Let $P = \{S_m^j\}$ be a ND-partitioning on G. For any nested dissection ordering α with respect to P, the number $\eta(G,\alpha)$ is bounded by

$$\sum_{m=0}^{h} \sum_{j=1}^{r_m} |S_m^j| \{|Adj(R_m^j)| + (|S_m^j| - 1)/2\}.$$

Proof: Note that $\eta(G,\alpha)$ is given by

$$\sum_{i=1}^{N} |Eadj(\alpha(i))|.$$

By theorem 3.1,

$$\sum_{\alpha(i) \in S_m^j} |Eadj(\alpha(i))|$$

$$\leq |Adj(R_m^j)| + (|S_m^j| - 1) + |Adj(R_m^j)| + (|S_m^j| - 2) + \ldots + |Adj(R_m^j)|$$

$$= \{|S_m^j| \cdot |Adj(R_m^j)| + |S_m^j| (|S_m^j| - 1)\},$$

and hence the result. □

Corollary 3.6 Let $P = \{S_m^j\}$ be a ND-partitioning with ordering α. Suppose further that P satisfies the following conditions, where $|X| = N$.

1) $h \leq C_1 \log_2 N$

2) $r_m \leq C_2 2^m$

3) $|S_m^j| \leq C_3 (N/2^m)^{1/2}$

4) $|Adj(R_m^j)| \leq C_4 |S_m^j|$.

Then $\eta(G,\alpha) \leq C_5 N \log_2 N$.

Proof: The bound in theorem 3.5 becomes

$$\sum_{m=0}^{h} \sum_{j=1}^{r_m} (C_4 + 1) C_3 (N/2^m)$$

$$\leq \sum_{m=0}^{h} C_2 2^m (C_4 + 1) C_3 (N/2^m)$$

$$\leq C_1 C_3 (C_4 + 1) C_2 N \log_2 N.$$ □

It should be noted that the same technique can be used to show that under the conditions in corollary 3.6, $\theta(G,\alpha) \leq O(N^{3/2})$.

The hypotheses in the corollary 3.6 are motivated by the problems arising in connection with two dimensional finite element meshes such as the one depicted in Figure 2.1, along with the observed behaviour of the algorithm we describe in section 5 when it is applied to such problems. Essentially, the algorithm is simply an implementation of the algorithmic definition of a nested dissection partitioning described in section 3.1. The algorithm is designed to choose the separators S_m^j so that they disconnect each R_m^j into (rarely more than) two components of approximately equal size. This immediately suggests hypotheses 1) and 2) of Corollary 3.6 and these do not depend upon the origin of the problem.

Now consider the implications of the graph being derived from a planar finite element mesh. If the mesh truly is two dimensional (rather than being "long and narrow"), then the number of boundary vertices X_B and the number of interior nodes X_I, should satisfy an "isoparametric" inequality of the form

$$K_1 X_B^2 \leq X_I \leq K_2 X_B^2, \tag{3.1}$$

for some constants K_1 and K_2. Now our algorithm is designed to find "small" separators S_m^j, which tends to generate components which are also two dimensional, and which therefore also satisfy inequalities of the form (3.1) above. This immediately suggests the plausibility of the hypotheses 3) and 4) in Corollary 3.6.

Similar reasoning for three dimensional meshes provides the motivation for corollary 3.7 below. The proof is again a simple application of theorem 3.5 and is omitted.

Corollary 3.7 Let the hypotheses of Corollary 3.6 apply except for condition 3), which is changed to

3) $$|S_m^j| \le C_6 \, (N/2^m)^{2/3}.$$

Then $$\eta(G,\alpha) \le C_7 \, N^{4/3}.$$

Similar arguments have recently been used by Eisenstat et. al. [7] to establish lower bounds of the same order as those in corollary 3.6 and 3.7. Thus orderings satisfying these hypotheses yield operation counts and fill which are asymptotically optimal.

In general, for a given ND-partitioning, there is a whole class of nested dissection orderings associated with it. In what follows, we show that many of them are equivalent in terms of storage and operation counts.

Theorem 3.8 Let α and β be two nested dissection orderings on $P = \{S_m^j\}$. If α conforms with β on P (see section 2 for the definition), then α and β are equivalent.

Proof: From the definition of $\theta(G,\alpha)$ and $\eta(G,\alpha)$, it is sufficient to show that $Eadj(x,\alpha) = Eadj(x,\beta)$ for all $x \in X$.

Let $x \in S_m^j$ and $\alpha(i) = x$, $\beta(k) = x$. Consider the subset R_m^j. Since both α and β numbers $R_m^j \backslash S_m^j$ before S_m^j, and they conform with each other on P, we have

$$R_m^j \cap \{\alpha(1),\ldots,\alpha(i)\} = R_m^j \cap \{\beta(1),\ldots,\beta(k)\}.$$

Furthermore, by lemma 3.2,

$$Adj(R_m^j) \cap \{\alpha(1),\ldots,\alpha(i)\} = \phi,$$

and $$Adj(R_m^j) \cap \{\beta(1),\ldots,\beta(k)\} = \phi.$$

Thus, from lemma 2.8, $Eadj(x,\alpha) = Eadj(x,\beta)$. □

Theorem 3.8 has the following important implication. In finding nested dissection orderings on P with small $\theta(G,\alpha)$ and $\eta(G,\alpha)$, it is sufficient to consider the subclass of nested dissection orderings which are compatible with P(that is, orderings which number nodes in S_m^j of P consecutively). Previously we noted that an ordering that numbers all the nodes of each partition member S_m^j before any of the nodes of its father in the dissection tree is a nested dissection ordering. Theorem 3.8 says we can restrict our attention to compatible orderings on the tree. The next section contains some conditions on P whereby all associated nested dissection orderings are equivalent.

§3.2 Minimal Nested Dissection Partitionings and Orderings

In this section, we consider a practically attractive subclass of ND-partitionings. A minimal nested dissection (MND) partitioning is a ND-partitioning $P = \{S_m^j\}$, where every separator S_m^j in $G(R_m^j)$ is minimal. The corresponding dissection orderings have the following interesting properties.

Lemma 3.9 Let $P = \{S_m^j\}$ be a MND-partitioning. For non-terminal member S_m^j, if $x, y \in S_m^j$, then y is reachable from x through $R_m^j \backslash S_m^j$.

Proof: Let $R_{m+1}^k \in C(R_m^j \backslash S_m^j)$. Since S_m^j is a minimal separator in $G(R_m^j)$, by lemma 2.1, $x,y \in Adj(R_{m+1}^k)$. But $G(R_{m+1}^k)$ is connected so that a path exists from x to y that goes through R_{m+1}^k. □

Corollary 3.10 Let P be a nested dissection ordering on a MND-partitioning P. If S_m^j is a non-terminal member, then $G(S_m^j)$ is a clique in the filled graph. □

Lemma 3.11 Let $P = \{S_m^j\}$be a MND-partitioning. If $Adj(R_m^j) \subset Adj(R_m^j \backslash S_m^j)$, then for $x \in S_m^j$ and $y \in Adj(R_m^j)$, y is reachable from x through $R_m^j \backslash S_m^j$.

□

Proof: It follows from the condition $Adj(R_m^j) \subset Adj(R_m^j \backslash S_m^j)$ and lemma 2.1. □

Theorem 3.12 Let $P = \{S_m^j\}$be a complete MND-partitioning satisfying $Adj(R_m^j) \subset Adj(R_m^j \backslash S_m^j)$ for all non-terminal member S_m^j. Then all nested dissection orderings on P are equivalent.

Proof: Let α be any nested dissection ordering on P, and consider those $S_m^j \neq R_m^j$. By lemma 3.11, $Adj(R_m^j) \subset Eadj(x,\alpha)$ for $x \in S_m^j$. Using lemma 3.9, we have

$$\sum_{x \in S_m^j} |Eadj(x,\alpha)|$$

$$= |Adj(R_m^j)| + (|S_m^j| - 1) + |Adj(R_m^j)| + (|S_m^j| - 2) + \ldots + |Adj(R_m^j)|,$$

and $$\sum_{x \in S_m^j} |Eadj(x,\alpha)|^2$$

$$= (|Adj(R_m^j)| + |S_m^j| - 1)^2 + (|Adj(R_m^j)| + |S_m^j| - 2)^2 + \ldots + |Adj(R_m^j)|^2,$$

which are independent of α. The above also holds for those $S_m^j = R_m^j$, since P is complete. This implies all nested dissection orderings on P are equivalent. □

It is interesting to note that under the conditions in the above theorem, the bound on $\eta(G,\alpha)$ is attained; that is,

$$\eta(G,\alpha) = \sum_m \sum_j |S_m^j| \{|Adj(R_m^j)| + (|S_m^j| - 1)/2\}.$$

While it is obviously possible for the condition $Adj(R_m^j) \subset Adj(R_m^j \backslash S_m^j)$ not to be satisfied, when the algorithm of section 5 is applied to our finite element problems we have found that the condition is very seldom violated. Even when it is, it is usually true that $|Adj(R_m^j)| - |Adj(R_m^j \backslash S_m^j)|$ is only one or two, so that the bound in Theorem 3.5 is quite tight for our problems. This implies that the storage scheme of section 4 will be very efficient; that is, very few zeros will be stored.

§4 The Storage Scheme for L and the Storage Allocation Algorithm

In this section we provide some of the more important aspects of the computer implementation of our linear equations solver. In particular, we describe the storage scheme used for the matrix factor L, along with the algorithm for performing the storage allocation, given the ordering α and the graph $G = (X,E)$. We provide this information before actually describing our ordering algorithm because the latter is motivated in part by the characteristics of our storage scheme.

Given the correspondence that was established in section 2, a partitioning P and a compatible ordering α of G specifies a particular ordering and partitioning of the matrix. Such matrix partitionings are obviously symmetric in the sense that the row and column partitionings are identical.

Let $p = |P|$, so that A is partitioned into p^2 submatrices A_{rs}, $1 \le r,s \le p$, and let L_{rs} be the corresponding submatrices of L, where $A = LL^T$. Let B_k be defined by

$$B_k = \begin{bmatrix} L_{k+1,k} \\ L_{k+2,k} \\ \vdots \\ L_{p,k} \end{bmatrix}, \qquad 1 \le k < p. \tag{5.1}$$

Obviously B_k is simply the part of the k-th block column which lies below $L_{k,k}$ in the partitioned matrix L.

The diagonal blocks of L are viewed as comprising a single block diagonal matrix whose rows, beginning at the first nonzero, are stored in a single one dimensional array ν.

An additional pointer vector δ of length N records the positions in ν of the diagonal components of each row, and another vector τ of length p records the beginning of each diagonal block in δ. For programming convenience we set $\tau_{p+1} = N + 1$, so that the size of the i-th partition member (diagonal block) is given by $\tau_{i+1} - \tau_i$, $1 \le i \le p$. This scheme is a minor variant of that proposed by Jennings [14].

As we shall see in section 5, the ordering of each member of P is done so that the non-null rows in each block column (B_k) are clustered together so that they form "blocks". These clusters of non-null rows within each block-column will be referred to as blocks, although it should be understood that they do not in general correspond to the row partitioning specified by P. For definiteness, we assume that there are μ such blocks.

The storage scheme which is naturally suggested and which we use is illustrated in Figure 4.1. The μ off-diagonal blocks are stored column by column in consecutive locations in a one-dimensional array ξ. The pointer vector γ points to the origins of these blocks in ξ, and the parallel vector ρ indicates the row number in L of the first row of each block. In order to specify which block column of L a particular block in ξ belongs to, we generate a vector σ of length p which points into γ so that the origins of the blocks in ξ contained in block-column i of L are given by γ_ℓ, $\sigma_i \le \ell < \sigma_{i+1}$. Again for programming convenience we set $\gamma_{\mu+1} = |\xi| + 1$ and $\sigma_p = \mu + 1$. Note that for $\sigma_i \le \ell < \sigma_{i+1}$, the number of rows in the block stored at ξ_{γ_ℓ} is given by $(\gamma_{\ell+1} - \gamma_\ell)/(\tau_{i+1} - \tau_i)$, so the row dimensions of the off-diagonal blocks do not have to be explicitly stored. The overhead storage obviously consists of that needed for the vectors δ, τ, γ, ρ, and σ, which is a total of $N + 2p + 2\mu + 3$ items. (See the remarks in section 6 about primary and overhead storage.)

It is interesting to note that our linear equations solver involves no computation with matrices stored in other than standard dense format. Apart from "driver" subroutines which determine the origins and dimensions of the blocks, all the subroutines operate on either dense matrices stored in standard columnby column format, or else in the now familiar envelope storage format due to Jennings [14].

We now describe our storage allocation algorithm. Let $P = \{Y_1, Y_2, \ldots, Y_p\}$ be the ordered partition of X induced by the compatible ordering α obtained by the algorithm of section 5. The algorithm used to determine the vectors σ, ρ and γ, which are necessary for the storage scheme, is given below. The vector τ is provided by the ordering algorithm, and the determination of δ is trivial.

1. Set $Z = \phi$ and $\sigma_1 = 1$.

2. For $i = 1,2,\ldots,p$ do the following:

   2a) Set $Z = Z \cup Y_i$

   2b) Determine the set $\tilde{C} = \mathrm{Adj}(C)$, where $G(C)$ is the component or components of $G(Z)$ containing Y_i.

   2c) Sort $\{\alpha^{-1}(v) \mid v \in \tilde{C}\}$ and determine the strings of consecutive integers; these correspond to the consecutive blocks within the block column of L corresponding to the partition member Y_i. This provides σ_{i+1} and the numbers ρ_ℓ and γ_ℓ, $\sigma_i \leq \ell < \sigma_{i+1}$.

§5 A Heuristic Algorithm for Automatic Dissection

The results of the section 3 suggest that what we should do is to recursively apply the following heuristic: find a "small" separator which disconnects the graph into two or more components of approximately equal size. The algorithm of this section is essentially an efficient implementation of this heuristic.

How do we find a small separator which disconnects a given graph into components of approximately equal size? Our strategy is to generate a level structure of the graph, rooted at a pseudo-peripheral node, and then to choose a minimal separator from a "middle" level. We root the level structure at a pseudo-peripheral node because this will tend to produce a structure with many levels, and the levels will tend to have relatively few nodes.

Given a graph $G = (X,E)$, the dissection algorithm is as follows.

1. Set $P = \phi$, $R = X$, and $N = |X|$.
2. If $R = \phi$, stop. Otherwise find a pseudo-peripheral node y in a connected component of $G(R)$, say $G(R_m^k)$.
3. Generate the rooted level structure $\mathcal{L}(y) = \{L_0, L_1, \ldots, L_\ell\}$ spanning $G(R_m^k)$. If $\ell \leq 1$, set $S_m^k = R_m^k$ and go to step 5. Otherwise set $j = \lfloor (\ell + 1)/2 \rfloor$.
4. Determine the set $S_m^k \subseteq L_j$ such that S_m^k is a minimal separator of $G(R_m^k)$.
5. Label S_m^k from $N - |S_m^k| + 1$ to N using the reverse Cuthill-McKee algorithm [15] applied to the subgraph $G(S_m^k)$. Set $N = N - |S_m^k|$.
6. Set $R = R \setminus S_m^k$, $P = P \cup \{S_m^k\}$, and go to step 2.

Step 4 in the dissection algorithm makes use of the following observation.

Lemma 5.1 Let $\mathcal{L}(x) = \{L_0, L_1, \ldots, L_{\ell(x)}\}$ be a rooted level structure at x. For $0 < j < \ell(x)$, the set

$$S = \{y \in L_j: \ \mathrm{Adj}(y) \cap L_{j+1} \neq \phi\} \qquad \square$$

is a minimal separator of the graph.

Thus, in step 4, we obtain S_m^k from L_j by simply discarding nodes in L_j which are not adjacent to any node in L_{j+1}.

It should be clear that the order in which the S_m^k are determined corresponds to a certain pre-order traversal of the dissection tree. That is, fathers are visited before their sons. Thus, we obtain a nested dissection ordering α compatible with P by labelling each S_m^k as it is determined, in decreasing order beginning with N.

Why label the S_m^k using the reverse Cuthill-McKee (RCM) algorithm, which is designed to produce a small bandwidth or profile? First, when $S_m^k = R_m^k$, so that S_m^k is a terminal node of the dissection tree, the envelope (or profile) of the diagonal block of the correspondingly ordered matrix will be preserved during the factorization. Since we use Jennings' storage scheme for the diagonal blocks, ordering these terminal members using the RCM algorithm will tend to reduce primary storage.

When S_m^k is a minimal separator, we showed in section 3 (corollary 3.10) that the corresponding diagonal block will completely fill in, so the ordering of S_m^k will have no effect on the primary storage required for the diagonal blocks of L corresponding to S_m^k. However, the way we number the S_m^k can substantially affect the amount of overhead storage required by the storage scheme we use in our linear equation solver, described in section 4.

In general, the nodes of S_m^k will be connected to some or all of the nodes in $Adj(R_m^k)$ in the filled graph $G^F = (X,E^F)$. Moreover, if $Adj(R_m^k) \neq \phi$, it must consist of one or more sets $\tilde{S}_\ell^j$, where $\tilde{S}_\ell^j \subseteq S_\ell^j$ and $\ell < m$.

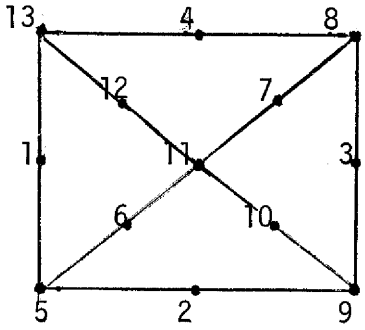
Now our storage scheme will be most efficient if the non-null rows within a block column (corresponding to S_m^k say) are clustered together, forming only a few blocks. It turns out that for our finite element applications, the S_ℓ^j's corresponding to separators typically consist of node sets lying on edges sequences of the finite element mesh. If these node sets are numbered consecutively, beginning at one end of the edge sequence, the non-null rows within each block column will be bunched together, as desired. Furthermore, our version of the RCM algorithm, which uses a pseudo-peripheral node as a starting node, will provide precisely this end-to-end ordering of the separator. Our implementation can be concisely described as follows:

Step 1: Determine a pseudo-peripheral node r and assign it to x_1.

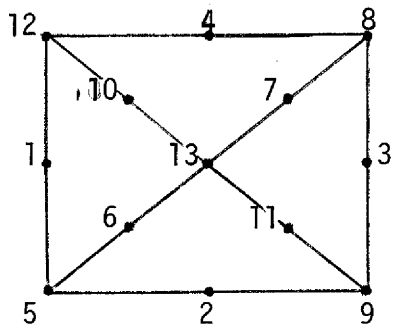
Step 2: For i = 1,2,...,N, find all the unnumbered neighbors of the node x_1 and number them in increasing order of degree.

Step 3: The RCM ordering is given by: $x_N, x_{N-1}, \ldots, x_1$.

The points above are illustrated by Figure 5.1, where the fill is the same for α_1 and α_2, but α_1 yields only 9 off-diagonal blocks compared to 15 for α_2. This difference is due to the different numbering of S_0. Since the overhead storage increases with the number of off-diagonal blocks, this clustering of the non-null off-diagonal rows in each block column reduces overhead storage, although as pointed out previously, it has no effect whatsoever on primary storage. The data structure for L is being used to select one of many equivalent nested dissection orderings compatible with $\mathcal{P}$.

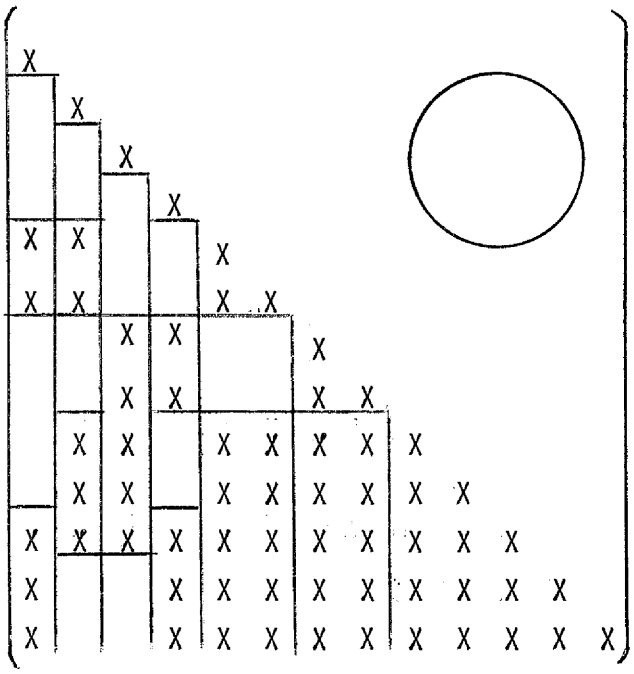


Finite element mesh with ordering α_1.

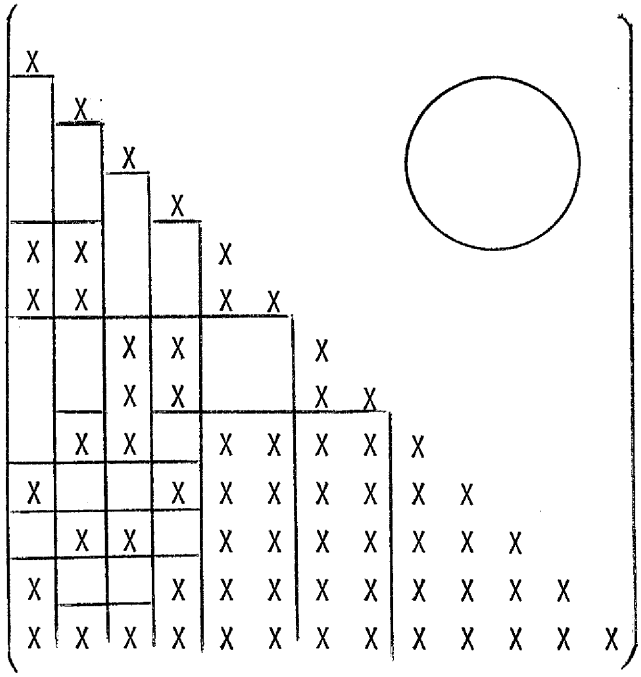


Finite element mesh with ordering α_2.

$S_0 = \{9,10,11,12,13\}$ $S_1^1 = \{5,6\}$, $S_1^2 = \{7,8\}$, $S_2^i = \{i\}$, $1 \le i \le 4$.



Matrix L corresponding to α_1.



Matrix L corresponding to α_2.

Figure 5.1 An example illustrating the effect of the ordering of the partition members on the number of off-diagonal blocks within each block column.

§6 Numerical Experiments

In this section we present some numerical experiments demonstrating the performance of our automatic nested dissection algorithm. Our first item of interest is to investigate the quality of the ordering produced by the algorithm when it is applied to the standard "benchmark" n by n grid problem, where $N = n^2$. The matrix problem corresponds in structure to that obtained when one applies a 9-point difference operator in connection with obtaining an approximate solution to an elliptic boundary value problem on a square domain. The figures in Table 6.1 compare the automatically produced ordering and the standard (explicitly prescribed) nested dissection ordering as described in [6,10].

N	Standard ordering from [10]	Standard ordering / N log N	Automatically produced ordering	Relative difference
100	1,010	2.19	1,072	.06
225	2,778	2.28	2,854	.03
400	5,988	2.50	6,443	.08
625	10,418	2.59	10,765	.03
900	16,434	2.69	17,127	.04
1225	24,096	2.77	25,006	.04

Table 6.1: Numbers of nonzeros in the triangular factor L for the standard nested dissection ordering of the n by n grid, compared to that for the automatically generated ordering.

The entries in Table 6.1 provide us with some reassurance that the ordering produced by our algorithm yields an L having approximately the same number of nonzeros as the factor corresponding to the standard ordering. Since it has been proven that for the standard ordering, the number of nonzeros in L is O(N log N), it seems safe to conclude that our automatically produced ordering has the same property. Note, however, how slowly the ratio to N log N converges. This is due to large negative subdominant terms in the expression for the number of nonzeros in L, as a function of N. See [10] for details.

Of course, the fact that our algorithm produces an efficient ordering for the n by n grid is of little practical interest, since it is a trivial exercise to write a program which will produce a nested dissection numbering of any rectangular grid. Our motivation in developing our algorithm was to enable us to find similarly efficient orderings for irregular mesh problems, where the required numbering is less obvious, and not so easily automated, even if it is known.

In order to gain some insight into the asymptotic behaviour of the algorithm and the execution times of the solver, we applied them to problems derived from the graded L mesh shown in Figure 6.1, subdivided by increasing subdivision factors s, yielding s^2 as many triangles as in the original mesh. The numerical results are reported in Tables 6.2 and 6.3.

In Table 6.2 we include both primary storage and overhead storage; that is, the storage used for actual numerical values and that used for pointers etc. associated with the matrix sparsity structure.

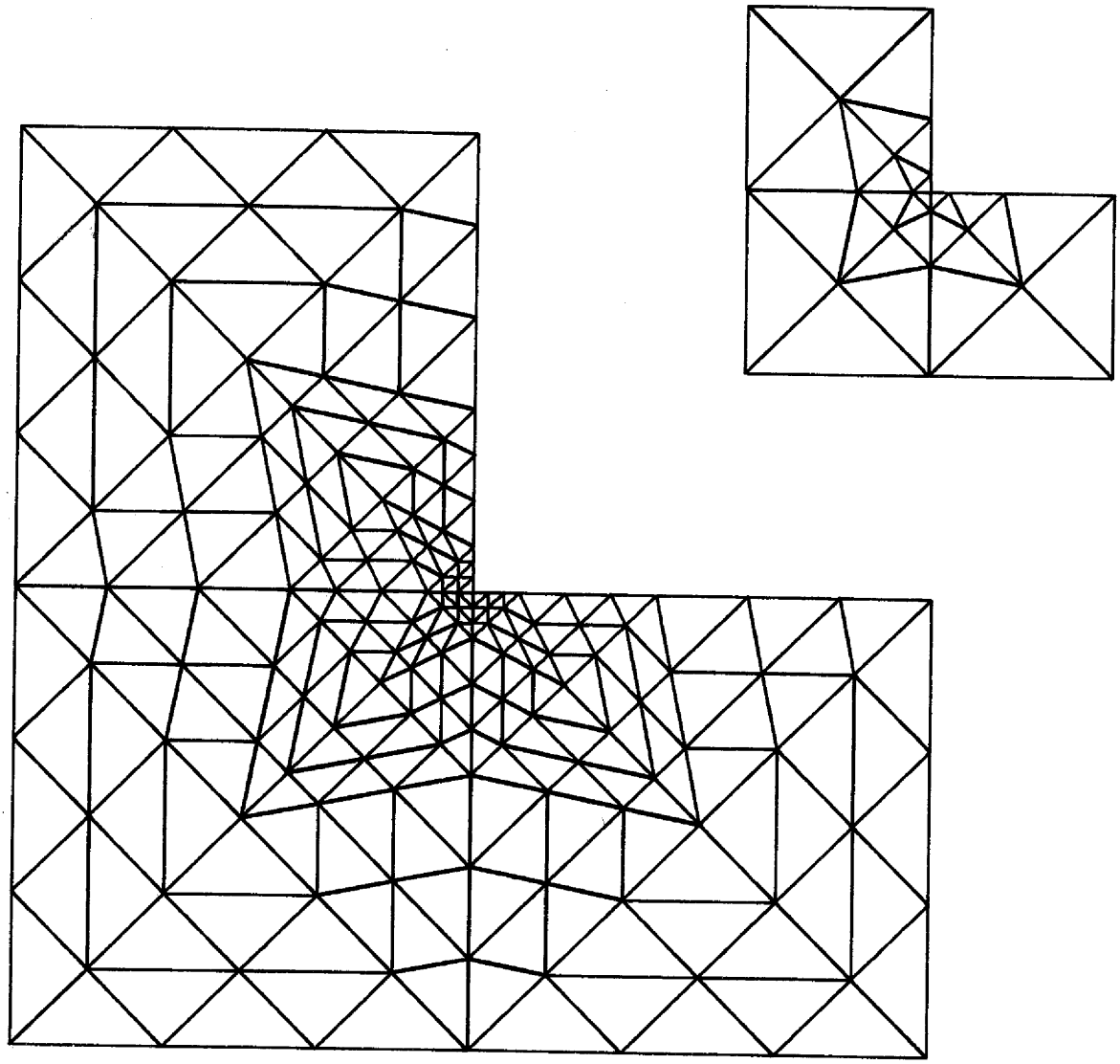


Figure 6.1: Graded L mesh.

Since one pays for storage regardless of the way it is employed, it is unfair in any practical study to ignore the overhead component. We do however distinguish between the two uses because on some computers having a large word size, it may be sensible to "pack" two or more pointers into each word. Our experiments were performed on an IBM 360/75 in single precision, and both pointers and the matrix entries were stored using 32 bits; however, by reporting primary and overhead storage separately, the reader can gauge their relative importance on other computers.

In our numerical experiments we also separate out 1) the execution time due to the actual determination of the ordering and data structure set-up, 2) the time required for the factorization of A into LL^T, and 3) the time required for the triangular solution given the factorization (i.e., solution of the triangular systems $Ly = b$ and $L^Tx = y$). We do this for two reasons. First, in some situations, many different problems having the same zero-nonzero structure must be solved. In this case, it may be quite reasonable to ignore the ordering and initialization time in comparing methods. Second, in the solution of some mildly nonlinear or time dependent problems, many systems having the same coefficient matrix but different right hand sides must be solved. In this context, the cost of solving the problem, given the ordering and the factorization, may be the primary factor in comparing methods.

In Table 6.3, operations mean multiplications and divisions. We report both execution times and operation counts to confirm that the operation counts are an accurate reflection of the amount of computation being performed. Since sparse matrix computations often involve rather sophisticated data structures, one must be assured that the computer time associated with data structure manipulation is at worst proportional to the amount of arithmetic being performed. Otherwise, comparing methods on the basis of operation counts makes no practical sense. All execution times are reported in seconds on an IBM 360/75 computer.

Table 6.2 Performance statistics on the ordering program, storage allocator, and data structure for the graded L mesh with s = 4(1) 14.

s	N	ORDER		STORAGE ALLOCATION		STORAGE				
		TIME	TIME / N log N	TIME	TIME / N log N	PRIMARY	PRIMARY / N log N	OVERHEAD	OVERHEAD / N	OVERHEAD / PRIMARY
4	265	.53	3.58(-4)	.20	1.35(-4)	3,692	2.50	1,207	4.55	.327
5	406	.86	3.52(-4)	.33	1.35(-4)	6,441	2.64	1,856	4.57	.288
6	577	1.27	3.46(-4)	.51	1.39(-4)	9,970	2.72	2,675	4.63	.268
7	778	1.76	3.40(-4)	.71	1.37(-4)	14,615	2.82	3,508	4.51	.240
8	1009	2.37	3.40(-4)	.97	1.40(-4)	20,100	2.88	4,555	4.51	.227
9	1270	3.06	3.37(-4)	1.28	1.41(-4)	26,798	2.96	5,892	4.64	.220
10	1561	3.86	3.36(-4)	1.63	1.42(-4)	34,468	3.00	7,365	4.72	.214
11	1882	4.72	3.32(-4)	2.01	1.42(-4)	43,298	3.05	8,804	4.68	.203
12	2233	5.69	3.30(-4)	2.45	1.42(-4)	53,155	3.09	10,611	4.75	.200
13	2614	6.77	3.29(-4)	2.91	1.41(-4)	64,614	3.14	12,074	4.62	.187
14	3025	7.94	3.28(-4)	3.42	1.41(-4)	76,883	3.17	13,731	4.54	.179

Table 6.3 Performance statistics for the linear equations solver for the graded L mesh with s = 4(1) 13.

s	N	FACTORIZATION								
		TIME	$\frac{\text{TIME}}{N^{3/2}}$	$\frac{\text{TIME}}{\text{OPERATIONS}}$	OPERATIONS	$\frac{\text{OPERATIONS}}{N^{3/2}}$	TIME	$\frac{\text{TIME}}{N \log N}$	$\frac{\text{TIME}}{\text{OPERATIONS}}$	OPERATIONS
4	265	.59	1.36(-4)	1.78(-5)	33,222	7.70	.16	1.08(-4)	2.16(-5)	7,382
5	406	1.08	1.32(-4)	1.57(-5)	68,810	8.41	.26	1.07(-4)	2.02(-5)	12,880
6	577	1.76	1.27(-4)	1.47(-5)	120,599	8.70	.38	1.04(-4)	1.91(-5)	19,938
7	778	2.73	1.26(-4)	1.37(-5)	199,544	9.20	.52	1.00(-4)	1.78(-5)	29,228
8	1009	3.94	1.23(-4)	1.31(-5)	301,146	9.40	.69	9.89(-4)	1.72(-5)	40,198
9	1270	5.50	1.21(-4)	1.24(-5)	441,586	9.76	.91	1.00(-4)	1.70(-5)	53,594
10	1561	7.45	1.21(-4)	1.21(-5)	612,683	9.93	1.15	1.00(-4)	1.67(-5)	68,934
11	1882	9.95	1.22(-4)	1.20(-5)	831,248	10.18	1.42	1.00(-4)	1.64(-5)	86,594
12	2233	12.51	1.18(-4)	1.15(-5)	1,085,835	10.30	1.70	.98(-4)	1.60(-5)	106,308
13	2614	15.59	1.16(-4)	1.11(-5)	1,404,174	10.50	2.00	.97(-4)	1.58(-5)	129,226

Several aspects of the information in Tables 6.2 and 6.3 are of interest:

1) The execution time of the ordering algorithm appears to be proportional to N log N. This is typical of such "divide and conquer" algorithms [1]. Unfortunately, we do not have a proof that the orderings we find always satisfy the hypotheses of Corollary 3.5. Such a proof would immediately provide a proof that the execution times of both the ordering algorithm and the storage allocator are O(N log N). Again, the results in Table 6.2 suggest that the execution time of the storage allocation is **O(N log N).**

2) **The storage <u>overhead</u> appears to grow <u>linearly</u> with N. This** contrasts with many sparse matrix storage schemes, where storage overhead is proportional to the number of nonzeros in L. Here it seems clear that overhead storage/primary storage $\to 0$ as $N \to \infty$.

3) Although we believe the operation counts for the factorization grows as $N^{3/2}$, it is difficult to be sure from the tables that this is indeed the case. Similarly, although it seems plausible from Table 6.1 that the primary storage (hence also the solution operation count), is proportional to N log N, the numbers in Table 6.2 are not completely convincing. We are reassured by the fact that the second column of Table 6.1 can be proven to grow as O(N log N), and its ratio to N log N also converges very slowly. The explanation appears to be the existence of very large subdominant terms in the expression for the operation and storage counts, as functions of N.

4) Even for problems of quite large size, the execution time required for the ordering algorithm is a significant portion of the overall time required to solve the problem. However, as mentioned earlier, there are circumstances where ordering time can be ignored because numerous problems with the same structure must be solved.

In practical terms, one is usually interested in comparing methods so as to determine the method which will yield the lowest computer charges. How does one measure cost? Clearly the real cost C is some function of execution time T and storage used S. Since the use of one method rather than another may simply increase T and reduce S, the function C(S,T) will be fundamental in determining which method results in the lowest charges. Computer memory continues to be a relatively expensive resource, so computer charging algorithms are usually designed to discourage large storage demands. Typically, C(S,T) is of the form $T \times p(S)$, where p is a polynomial of degree ≥ 1. Thus, we contend that storage reduction should be regarded as at least as important as reduction in execution time. It is frequently the case that a method which doubles the execution time and halves the storage requirements of another method yields a net reduction in computer charges. Another important related point is that a reduced storage requirement may allow one to avoid using auxiliary storage. The value of this is hard to assess quantitatively, but the cost of data transfer can be substantial.

In order to gain some insight into how our new ordering algorithm/solver combination compares to more standard approaches, we solved the sequence of graded L problems using the reverse Cuthill-McKee ordering algorithm (RCM) [15] and a standard linear equations solver using the envelope storage scheme due to Jennings. A large amount of experience with finite element problems has established this combination as the industry standard.

For purposes of comparison, we assumed a computer charging function which is proportional to the product of execution time and the storage used. The comparison is summarized in Table 6.4, where "envelope" refers to the combination of the RCM ordering algorithm and the linear equation solver using Jennings' envelope storage scheme, and "dissection" refers to our new ordering algorithm/solver.

As a one-shot method for solving the problem, the new scheme does not compare favorably to the standard method until $N \tilde{\sim} 1500$. This is because the execution times for the ordering algorithm and the data structure initialization are substantially larger than those for the RCM algorithm and the data structure set up associated with the envelope scheme. The storage requirement for the new algorithm is also slightly higher than for the RCM algorithm.

In those situations where the initial cost of finding the ordering and initializing the data structures can be ignored because many different problems having the same structure must be solved, the cross-over point drops to $N \tilde{\sim} 1200$.

Finally, in cases where many problems differing only in their right hand sides must be solved, the cross-over point does not occur until $N \tilde{\sim} 2300$. The theoretical cross-over point (based on operation count rather than execution time) is much lower, but the increased data structure complexity of the new scheme with the consequent execution overhead makes it less efficient than the standard scheme. The same remarks are true for the factorization, but the overhead associated with accessing each block is not so important there because much more computation is performed on each block.

Similar results to those in Table 6.4 have been obtained for numerous other mesh examples, although as one would expect, the actual cross-over points depend on the mesh characteristics such as whether it has appendages, holes, etc. The results in Table 6.4 are, however, quite representative.

Table 6.4 Comparison of envelope and dissection schemes, assuming cost is proportional to storage x execution time. The problem is the graded-L mesh with s = 7(1) 12. The numbers in the table are cost (dissection)/ cost (envelope).

s	N	Ordering, storage allocation, factorization & solution	Factorization and solution	Solution, given the factorization
7	778	1.66	1.31	1.80
8	1009	1.40	1.10	1.55
9	1270	1.22	.97	1.39
10	1561	1.04	.84	1.27
11	1882	.93	.76	1.13
12	2233	.81	.66	1.01

§7 Concluding Remarks

We have formally defined nested dissection partitionings and orderings of an undirected graph. We have also introduced the notion of elimination-adjacent set Eadj(x) of a node x in a labelled graph G, which is important because it provides a direct relationship between the structures of a matrix and its Cholesky factors. We then provided some results about the fill suffered by matrices whose graphs are labelled by a nested dissection ordering.

We then described an automatic scheme for finding nested dissection partitionings and orderings. We also described a data structure for Cholesky factors of matrices thusly ordered which facilitates the design of an efficient linear equations solver.

Finally, we applied this ordering algorithm/linear equations solver to some finite element problems, obtaining some numerical results which suggest that for these problems that:

1) the execution time of the ordering algorithm is O(N log N).
2) the number of nonzeros in the Cholesky factor for the ordering produced is O(N log N).
3) the operation count for the factorization is $O(N^{3/2})$.
4) the overhead storage required for the data structure for L is O(N).
5) under some reasonable assumptions about computer charges, our ordering/solution package appears to be preferable to the standard band oriented approach provided N is reasonably large.

§8 References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. (1974).

[2] I. Arany, W.F. Smyth and L. Szoda, "An improved method for reducing the bandwidth of sparse symmetric matrices", in Information Processing 71: Proceedings of IFIP Congress, North-Holland, Amsterdam, (1972).

[3] C. Berge, The Theory of Graphs and its Applications, John Wiley & Sons Inc., New York, (1962).

[4] H.L. Crane, Jr., N.E. Gibbs, W. G. Poole, Jr., and P.K. Stockmeyer, "Matrix bandwidth and profile minimization", Report No. 75 - 9, ICASE Report (1975).

[5] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices", Proc. 24th Nat. Conf., Assoc. Comput. Mach., ACM Publ. P - 69, 1122 Ave. of the Americas, New York, N.Y. (1969).

[6] I.S. Duff, A.M. Erisman and J. K. Reid, "On George's nested dissection algorithm", SIAM J. Numer. Anal., to appear.

[7] S.C. Eisenstat, M.H. Schultz, and A. H. Sherman, "Applications of an element model for Gaussian elimination", in Sparse Matrix Computations, edited by J.R. Bunch and D.J. Rose, Academic Press, (1976).

[8] G.C. Everstine, "The BANDIT computer program for the reduction of matrix bandwidth for NASTRAN", NSRDC Report 3827 (1972).

[9] Alan George, "Nested dissection of a regular finite element mesh", SIAM J. Numer. Anal., 10 (1973), pp. 345 - 363.

[10] Alan George, "Numerical experiments using dissection methods to solve n by n grid problems", SIAM J. Numer. Anal., to appear.

[11] Alan George and Joseph W.H. Liu, "An implementation of a pseudo-peripheral node finder", Dept. of Computer Science Technical Report CS-76-44, University of Waterloo, October, (1976).

[12] Alan George and Joseph W.H. Liu, "An algorithm for automatic nested dissection and its application to general finite element problems", Proc. Sixth Conference on Numerical Mathematics and Computing, Winnipeg, Manitoba, Sept.30 - Oct. 2, (1976).

[13] N.E. Gibbs, W.G. Poole and P.K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix", SIAM J. Numer. Anal., 13 (1976), pp. 236 - 250.

[14] A. Jennings, "A compact storage scheme for the solution of simultaneous equations", Comput. J., 9 (1966), pp. 281 - 285.

[15] Joseph W.H. Liu and Andrew H. Sherman, "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices", SIAM J. Numer. Anal., 13 (1976), pp. 198 - 213.

[16] S.V. Parter, "The use of linear graphs in Gauss elimination". SIAM Rev., 3 (1961), pp. 364 - 369.

[17] D.J. Rose, "A graph theoretic study of the numerical solution of sparse positive definite systems", in Graph Theory and Computing, R.C. Read, editor, Academic Press,(1972).

[18] D.J. Rose and G. F. Whitten, "A recursive analysis of dissection strategies", in Sparse Matrix Computations, edited by D.J. Rose and J.R. Bunch, Academic Press, (1976).

[19] O.C. Zienkiewicz, The Finite Element Method in Engineering Science, McGraw-Hill, London (1970).