# A MINIMAL STORAGE IMPLEMENTATION OF THE MINIMUM DEGREE ALGORITHM†

by

Alan George*

and

Joseph W.H. Liu**

Research Report CS-77-09

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

* 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

We describe an efficient implementation of the minimum degree algorithm, which experience has shown to be effective in finding low fill orderings for sparse positive definite systems. The algorithm is heuristic; at each step in the elimination the variable chosen to eliminate next is that which minimizes the fill suffered at that step. Thus, some representation of the partially factored matrix is required at each step of the ordering. Previous implementations have stored this representation in an explicit form, which requires a data structure which allows the matrix structure to change as the ordering proceeds. The implementation we describe in this paper works only with the graph of the original matrix, and all data structures used are fixed throughout the execution of the algorithm. In contrast to most previous implementations, the total storage needs of the algorithm are known before execution. Several effective techniques for speeding up the algorithm are described, and numerical experiments on some problems arising in finite element applications suggest that for these problems the execution time is O(N), where N is the number of equations.

## Introduction

Consider the symmetric positive definite system of linear equations

(1.1) $Ax = b$,

where the N by N matrix A is sparse. If we solve (1.1) using Cholesky's method, A is first factored into $LL^T$, where L is lower triangular, and then we solve the triangular systems $Ly = b$ and $L^Tx = y$. Sparse matrices normally suffer some fill when they are factored, so $L + L^T$ is usually fuller than A. Since for any N by N permutation P, the matrix $PAP^T$ is still symmetric and positive definite, we can still use Cholesky's method to solve the equivalent problem

(1.2) $(PAP^T)(Px) = Pb$.

A judicious choice of P can drastically reduce fill, hence the interest in algorithms for finding such permutations.

A heuristic algorithm which experience has shown to be extremely effective in finding low-fill orderings is the so-called minimum degree algorithm [7]. It is a "local" algorithm which at each elimination step permutes the part of the matrix remaining to be factored so that a column (row) with the fewest nonzeros is in the pivot position. This implies that at each step of the algorithm we need a representation of the structure of the partially factored matrix. Previous implementations known to the authors [1, 3, 8] store some explicit representation of the partially factored matrix, which has two disadvantages. Since the structure changes as the elimination (or simulation thereof) proceeds, the data structure must be flexible enough to allow for such changes. Second, it is usually impossible to predict the maximum storage requirements for such implementations;

usually the storage requirement grows for a time and then tapers off near the conclusion of the ordering.

The implementation we describe in this paper operates only on the original matrix graph. The data structures remain fixed during the execution of the algorithm, and storage requirements consist of a small number of vectors of length N together with the storage needed for the graph.

An outline of the paper is as follows. In section 2 we introduce some graph theory notions which are required in describing the algorithm and its implementation. Section 3 contains a description of the basic algorithm, and section 4 contains some crucial refinements to the algorithm which appears to reduce its time complexity to O(N) for finite element problems. Section 5 contains some programming details along with some numerical experiments, and section 6 contains our conclusions.

## §2 Preliminaries

In this section we review some basic graph theory notions that are related to symmetric Gaussian elimination. We also consider graph theoretic ways of viewing the elimination process.

### 2.1 Some Graph Theory Terminology

A graph $G = (X,E)$ consists of a finite nonempty set $X$ of nodes together with a prescribed edge set $E$ of unordered pairs of distinct nodes. A graph $G' = (X',E')$ is a subgraph of $G = (X,E)$ if $X' \subset X$ and $E' \subset E$. For $Y \subset X$, $G(Y)$ refers to the subgraph $(Y,E(Y))$ of $G$, where $E(Y) = \{\{u,v\} \in E | u,v \in Y\}$.

Nodes $x$ and $y$ are said to be adjacent if $\{x,y\}$ is an edge in $E$. For a subset $Y$ of nodes, the adjacent set of $Y$ is defined as

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

If $Y = \{y\}$, we shall write $\mathrm{Adj}(y)$ instead of the formally correct $\mathrm{Adj}(\{y\})$. The degree of a node $x$ is the number of nodes adjacent to $x$, denoted by $|\mathrm{Adj}(x)|$. Sometimes, we shall refer to $y \in \mathrm{Adj}(x)$ as a neighbor of the node $x$.

A path of length $\ell$ is an ordered set of distinct nodes $(v_0,v_1,\ldots,v_\ell)$ where $v_i \in \mathrm{Adj}(v_{i-1})$, $1 \le i \le \ell$. A graph $G$ is connected if there is a path connecting each pair of distinct nodes. If $G$ is disconnected, it consists of two or more maximal connected subgraphs called components.

Let $Y$ be a subset of the node set $X$. The component partitioning $\mathcal{C}(Y)$ of $Y$ is defined as:

$$\mathcal{C}(Y) = \{Y' \subset Y | G(Y') \text{ is a connected component in the subgraph } G(Y)\}.$$

When $Y = X$, $\mathcal{C}(X)$ is simply the set of component sets in the graph $G$.

A useful notion in the study of Gaussian elimination is the <u>reachable set</u> which we now define. Let $S$ be a subset of $X$ and $y \in X\backslash S$. The node $y$ is said to be <u>reachable from</u> a node $x$ <u>through</u> $S$ if there exists a path $(x, v_1, \ldots, v_k, y)$ such that $v_i \in S$, for $1 \leq i \leq k$. Note that $k$ can be zero, so that any adjacent node of $y$ not in $S$ is reachable from $y$ through $S$.

**The <u>reachable set of</u> $y$ <u>through</u> $S$, denoted by Reach $(y,S)$, is then defined to be**

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

We can extend this definition to subsets of $X$. Let $Y \subset X$ with $Y \cap S = \phi$. The reachable set of $Y$ through $S$ is then

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

Note that $\text{Adj}(Y)\backslash S \subset \text{Reach }(Y,S)$. When $S = \phi$, it can be seen that

$$\text{Adj}(Y) = \text{Reach }(Y,\phi),$$

so that the reachable set concept may be regarded as a generalization of the adjacent set.

## §2.2 The Elimination Process in terms of Elimination Graphs

In this section, we review the graph theory approach used by Parter [5] and Rose [7] to study the Gaussian elimination process. We first establish a correspondence between graphs and matrices.

Let A be an N by N symmetric metrix. 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:

$$X^A = \{x_1, \dots, x_n\},$$

and $\{x_i, x_j\} \in E^A$ if and only if $A_{ij} \neq 0$. For any N by N permutation matrix P, the unlabelled graphs of A and $PAP^T$ are the same, but the associated labellings differ.

Consider the symmetric factorization of the matrix A into $LL^T$. The elimination process applied to A can be interpreted as a sequence of graph transformations on $G^A$. Following Rose [7], we define an elimination graph as follows. Let G = (X,E) be a graph and y be a node in G. The <u>elimination graph</u> of G by y, denoted by $G_y$, is the graph

$$(X \backslash \{y\},\ E(X \backslash \{y\}) \cup \{\{u,v\} \mid u,v \in Adj(y)\}).$$

With this definition, the process of Gaussian elimination on a matrix A can be viewed as a sequence of elimination graphs

$$G_0, G_1, \dots, G_{N-1}$$

where $G_0 = G^A$,

and $G_i = (G_{i-1})_{x_i} = (X_i, E_i)$.

Here the set of nodes $X_i = \{x_{i+1}, \dots, x_N\}$. The graph $G_i$ precisely reflects the structure of the matrix remaining to be factored after the i-th step of the Gaussian elimination. This interpretation provides insight into the elimination process, and it is useful in the study of the fill-in phenomenon.

We now relate properties of the triangular factor L of A with the elimination graph sequence. In the factor L, let

$$\nu_i = |\{L_{ji} : L_{ji} \neq 0, j > i\}| \quad \text{for } i = 1,\ldots,N.$$

<u>Lemma 2.1</u> [7] The quantity $\nu_i$ is the degree of the node $x_i$ in the elimination graph $G_{i-1}$. □

Since the number of off-diagonal nonzeros in the factor L is given by

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

and the number of multiplicative operations required for the factorization is

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

the degree of the node $x_i$ in $G_{i-1}$ plays an important role in the storage and operation requirements of the elimination process. The minimum degree algorithm to be studied in section 3 is designed to reduce these requirements by a local minimization of the degrees.

## §2.3 An Alternative View using Reachable Sets

In section 2.2, the quantities $\nu_i$ in the factor L are related to node degrees in the sequence of elimination graphs. In this section, we provide a direct relation between these numbers $\nu_i$ and the original graph $G^A$ associated with the matrix A. This relationship has been established elsewhere but we include it here for completeness. The approach uses the notion of reachable sets introduced in section 2.1.

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 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 [ 5].

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 $k < \min\{i,j\}$. □

In order to relate the numbers $\nu_i$ to the graph $G^A$, we characterize the edge set $E^F$ using only $G^A$. The following results are quoted from [2]. Lemma 2.2 is equivalent to Lemma 4 in [8].

Lemma 2.2 Let $j > i$. The unordered pair $\{x_i,x_j\} \in E^F$ if and only if $x_j \in \text{Reach}\,(x_i,\{x_1,\ldots,x_{i-1}\})$. □

Corollary 2.3 For $i = 1,\ldots,N$,

$$\nu_i = |\text{Reach}\,(x_i,\{x_1,\ldots,x_{i-1}\})|.$$ □

Not only does the reachable set concept characterize the qualities $\nu_i$, it also reflects the adjacency structure of the elimination graph $G_i$. Let $G_0, G_1, \ldots, G_{N-1}$ be the sequence of elimination graphs as defined by the nodes $x_1$, $x_2, \ldots, x_N$. The next lemma follows from the definitions of elimination graphs and reachable sets, and it can be proved by induction.

<u>Lemma 2.4</u> Let $y$ be any node in the elimination graph $G_i = (X_i, E_i)$. The set of adjacent nodes of $y$ <u>in $G_i$</u> is given by

$$\text{Reach}\ (y, \{x_1, \ldots, x_i\}),$$

where the Reach operator is taken in the <u>original graph</u>. □

**The above lemma is useful in the next section when the minimum degree algorithm is studied.**

## §3 The Minimum Degree Algorithm

### §3.1 Description of the Algorithm Using Elimination Graphs

Following Rose [ 7 ], we describe the minimum degree algorithm using elimination graphs. Let $G_0 = (X,E)$ be an unlabelled graph.

Step 1 (Initialization) $i \leftarrow 1$

Step 2 (Minimum degree selection) In the elimination graph $G_{i-1}$, choose $x_i$ to be a node such that

$$|\mathrm{Adj}(x_i)| = \min_{y \in X_{i-1}} |\mathrm{Adj}(y)|,$$

where $G_{i-1} = (X_{i-1}, E_{i-1})$.

Step 3 (Graph transformation) Form the new elimination graph $G_i = (G_{i-1})_{x_i}$.

Step 4 (Loop or stop) $i \leftarrow i + 1$. If $i > |X|$, stop. Otherwise, go to step 2.

The above formulation of the algorithm involves the formation of the sequence of elimination graphs. An implementation of this description can be found in the Yale Sparse Matrix Package (Sherman [9]).

## §3.2 Description of the Algorithm using Reachable Sets

In the description of the minimum degree algorithm in section 3.1, the sole purpose of step 3,the graph transformation,is to facilitate the selection of the next node from the new elimination graph. This step can be omitted if we can provide an alternate way to compute the degrees of the nodes in the elimination graph. Lemma 2.4 shows that the reachable set is the relevant concept to use.

The lemma relates the adjacency structure of the elimination graphs to that of the original graph. With this simple connection, we can restate the minimum degree algorithm in terms of reachable sets.

```
Step 1   (Initialization)     $S \leftarrow \phi$.
         $DEG(x) \leftarrow |Adj(x)|$, for $x \in X$.
Step 2   (Minimum degree selection) Pick a node $y \in X \backslash S$
         where $DEG(y) = \min_{x \in X \backslash S} DEG(x)$.

         Number the node y next and set $S \leftarrow S \cup \{y\}$.

Step 3   (Degree update)
         $DEG(x) \leftarrow |Reach(x,S)|$ for $x \in X \backslash S$.
Step 4   (Loop or stop). If $S = X$, stop. Otherwise, go to step 2.
```

## §3.3 Some Related Results on Reachable Sets

The observation in section 3.2 shows that the reachable sets deserve more detailed analysis. In this subsections we establish some preliminary results in this direction.

Consider a graph $G = (X,E)$. Let $S$ be a (possibly empty) subset of $X$ and $C(S)$ be the component partitioning of $S$. (See section 2 for the definition).

Consider a node $y$ in $X \backslash S$. Let

$$\{C_1, \dots, C_k\} \subset C(S)$$

be all the connected components in $G(S)$ with $y \in \mathrm{Adj}(C_i)$, $1 \le i \le k$. Note that the number $k$ depends on the subset $S$ and the node $y$. Define the <u>neighborhood of</u> $y$ <u>in</u> $S$ to be the subset

$$\mathrm{Nbrhd}(y,S) = \bigcup_{i=1}^{k} C_i .$$

The following lemmas relate neighborhoods to reachable sets.

<u>Lemma 3.1</u> $\mathrm{Nbrhd}(y,S) = \bigcup_{T \subset S} \mathrm{Reach}\,(y,T) \cap S.$ □

In other words, it can be expressed as:

$$\mathrm{Nbrhd}(y,S) = \{s \in S \mid s \text{ is reachable from } y \text{ through a subset of } S\}.$$

**Lemma 3.2** Reach $(y,S) = \mathrm{Adj}(\mathrm{Nbrhd}(y,S) \cup \{y\})$.

Proof Consider $u \in$ Reach $(y,S)$. There exists a path $(u,s_1,\ldots,s_t,y)$ where $s_i \in S$, $1 \le i \le t$. If $t = 0$, $u \in \mathrm{Adj}(y)$ so that $u \in \mathrm{Adj}(\mathrm{Nbrhd}(y,S) \cup \{y\})$. If $t \neq 0$, we have $s_1 \in \mathrm{Nbrhd}(y,S)$ and thus $u \in \mathrm{Adj}(\mathrm{Nbrhd}(y,S))$.

On the other hand, let $v \in \mathrm{Adj}(\mathrm{Nbrhd}(y,S) \cup \{y\})$. Either $v \in \mathrm{Adj}(y) \setminus S$ or $v \in \mathrm{Adj}(s)$ for some $s \in \mathrm{Nbrhd}(y,S)$. In both cases, $v$ is reachable from $y$ through $S$. □

For convenience in later discussions, we introduce one more definition. The closure of $y$ by $S$ is defined by

$$\mathrm{Closure}\ (y,S) = \mathrm{Nbrhd}\ (y,S) \cup \{y\} \cup \ \mathrm{Reach}\ (y,S).$$

It is clear that the closure is a disjoint union of

$$\mathrm{Nbrhd}\ (y,S) \subset S$$

and $$\mathrm{Reach}\ (y,S) \cup \{y\} \subset X\setminus S.$$

It is interesting to point out that "Reach", "Nbrhd", and "Closure" may be regarded as operators:

$$\begin{aligned} &\mathrm{Reach}: && X\setminus S \to P(X\setminus S) \\ &\mathrm{Nbrhd}: && X\setminus S \to P(S) \\ &\mathrm{Closure}: && X\setminus S \to P(X), \end{aligned}$$

where $P(*)$ is the power set of the specified set.

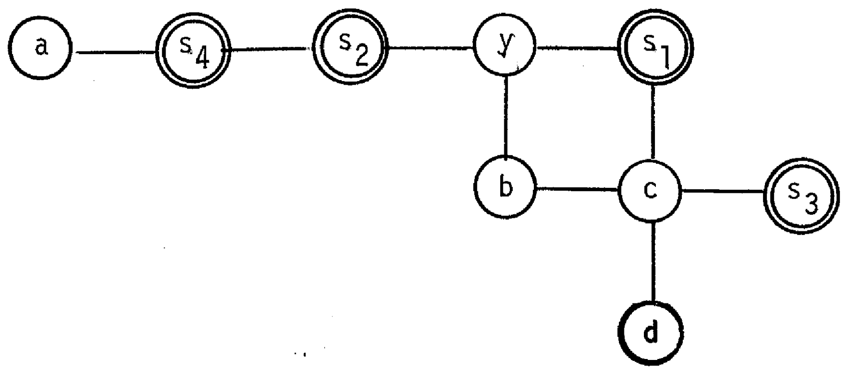


Figure 1. A 9-node graph.

The definitions and results can be best illustrated by an example. Consider the graph in Figure 1. Let $S = \{s_1,s_2,s_3,s_4\}$ and $y \in S$. It can be seen that the corresponding value of k is two, with

$$C_1 = \{s_1\},$$

and

$$C_2 = \{s_2,s_4\}.$$

So, the neighborhood set is

$$Nbrhd(y,S) = \{s_1,s_2,s_4\}.$$

By lemma 3.2, we have

$$\begin{aligned} Reach(y,S) &= Adj(\{s_1,s_2,s_4,y\}) \\ &= \{a,b,c\}. \end{aligned}$$

The closure set is then

$$Closure\ (y,S) = \{s_1,s_2,s_4,y,a,b,c\}.$$

The following results contain observations that are crucial to our implementation of the minimum degree algorithm to be discussed later in the paper.

Lemma 3.3 Let $x \in X \backslash S$. If

$\mathrm{Adj}(x) \subset \mathrm{Closure}(y,S)$

then $\mathrm{Nbrhd}(x,S) \subset \mathrm{Nbrhd}(y,S)$.

Proof Consider any $s \in \mathrm{Nbrhd}(x,S)$. There exists a path $x,s_1,\ldots,s_t,s$, where $s_i \in S$ for $1 \le i \le t$. If $t = 0$, $s \in \mathrm{Adj}(x) \cap S \subset \mathrm{Closure}(y,S) \cap S = \mathrm{Nbrhd}(y,S)$. If $t \ne 0$, $s_1$ will belong to $\mathrm{Nbrhd}(y,S)$ so that $s \in \mathrm{Nbrhd}(y,S)$. □

Theorem 3.4 Let $x \in X \backslash S$. If

$\mathrm{Adj}(x) \subset \mathrm{Closure}(y,S)$

then $\mathrm{Reach}(x,S) \subset \mathrm{Reach}(y,S) \cup \{y\}$.

Proof Consider a node $w \in \mathrm{Reach}(x,S)$ and $w \ne y$. There exists a path $x,s_1,\ldots,s_t,w$, where $s_i \in S$ for $1 \le i \le t$.

If $t = 0$, $w \in \mathrm{Adj}(x) \cap (X \backslash S) \subset \mathrm{Closure}(y,S) \cap (X \backslash S)$, so that $w \in \mathrm{Reach}(y,S)$. Otherwise, if $t \ne 0$, $s_1 \in \mathrm{Adj}(x) \cap S \subset \mathrm{Closure}(y,S) \cap S$ $= \mathrm{Nbrhd}(y,S)$. So a path in $S$ can be traced from $w$ to $y$; in other words $w \in \mathrm{Reach}(y,S)$. □

Corollary 3.5 Let $x$ be as in theorem 3.4. Then $|\mathrm{Reach}(x,S)| \le |\mathrm{Reach}(y,S)|$.

Proof If $y \notin \mathrm{Reach}(x,S)$, by theorem 3.4

$\mathrm{Reach}(x,S) \subset \mathrm{Reach}(y,S)$,

so that the result follows. On the other hand, assume $y \in \mathrm{Reach}(x,S)$. Then $x \in \mathrm{Reach}(y,S)$ and we have $\mathrm{Reach}(x,S) \cup \{x\} \subset \mathrm{Reach}(y,S) \cup \{y\}$.

Thus, $|\mathrm{Reach}(x,S)| = |\mathrm{Reach}(x,S) \cup \{x\}| - 1$

$$\le |\mathrm{Reach}(y,S) \cup \{y\}| - 1$$

$$= |\mathrm{Reach}(y,S)|. \qquad \square$$

It is instructive to illustrate theorem 3.4 with an example. In the example of figure 1, we have noted that

$$\mathrm{Nbrhd}(y,S) = \{s_1,s_2,s_4\}$$

and

$$\mathrm{Reach}(y,S) = \{a,b,c\}.$$

The nodes a, b, d satisfy the condition in theorem 3.4 since

$$\mathrm{Adj}(a) = \{s_4\}$$

$$\mathrm{Adj}(b) = \{y,c\}$$

$$\mathrm{Adj}(d) = \{c\}.$$

However, the node c fails.

The definitions introduced in this subsection can be extended to nonempty subsets. For any nonempty subset $Y \subset X \backslash S$, the readers are left to formalize the notions Nbrhd(Y,S) and Closure (Y,S).

## §4 Refinements of the Algorithm using Reachable Sets

In this section, we consider novel features in our implementation of the minimum degree algorithm as described in section 3.2. It should be emphasized that we only operate on the adjacency structure of the original graph. This has the obvious advantage of keeping the original graph structure intact. In addition, no complicated data structure is necessary to allow for graph transformations.

### §4.1 Minimum Degree Selection

For each execution of step 2 of the algorithm described in section3.2, a node of minimum degree is selected and numbered. In what follows, we show that it may be possible to number a set of nodes at one time and that the amount of extra work involved is small. This basic observation has already been made and utilized elsewhere [3,9]. However, its previous application used information which is not available to us directly, since we do not have an explicit representation of the partially eliminated matrix. Our objective here is to establish conditions whereby we can carry out a "mass elimination", in terms of information provided by the REACH operator.

$$\text{Let } y \in X\backslash S \text{ satisfying } |\mathrm{Reach}(y,S)| = \min_{z \notin S} |\mathrm{Reach}(z,S)|.$$

Theorem 4.1 Let $x \in X\backslash S$ and

(4.1) $\mathrm{Adj}(x) \subset \mathrm{Closure}(y,S)$.

Then $|\mathrm{Reach}(x,S)| = |\mathrm{Reach}(y,S)|$.

Proof From the choice of the node y, we have

$|\mathrm{Reach}(y,S)| \leq |\mathrm{Reach}(x,S)|$.

The result then follows from corollary 3.5. □

Among those nodes satisfying the adjacency condition in theorem 4.1, we restrict ourselves only to those in $\mathrm{Reach}(y,S) \cup \{y\}$. To this end, we define the set

(4.2) $Y = \{x \in \{\mathrm{Reach}(y,S) \cup \{y\} \mid \mathrm{Adj}(x) \subset \mathrm{Closure}(y,S)\}$.

We now establish some properties of the set $Y$.

Lemma 4.2 Let $x \in Y$. Then

$$\mathrm{Reach}(x,S) \cup \{x\} = \mathrm{Reach}(y,S) \cup \{y\}.$$

Proof Assume $x \neq y$. Then $x \in Y \backslash \{y\} \subset \mathrm{Reach}(y,S)$. Together with theorem 3.4, we have

$$\mathrm{Reach}(x,S) \cup \{x\} \subset \mathrm{Reach}(y,S) \cup \{y\}.$$

By theorem 4.1, the two sets must be the same. □

We can generalize the result of lemma 4.2 to an arbitrary subset of $Y$. Let $Y'$ be a non-empty subset of $Y$.

Lemma 4.3 $\mathrm{Reach}(Y',S) \cup Y' = \mathrm{Reach}(y,S) \cup \{y\}$.

Proof By definition, $Y' \subset Y \subset \mathrm{Reach}(y,S) \cup \{y\}$. Note also that $\mathrm{Reach}(Y',S) = (\bigcup_{x \in Y'} \mathrm{Reach}(x,S)) \backslash Y'$, so that by theorem 3.4 we have

$$\mathrm{Reach}(Y',S) \subset \bigcup_{x \in Y'} \mathrm{Reach}(x,S) \subset \mathrm{Reach}(y,S) \cup \{y\}.$$

On the other hand, clearly $y \in \mathrm{Reach}(Y',S) \cup Y'$. Let $v \in \mathrm{Reach}(y,S)$ and $v \notin Y'$. Pick any $x \in Y'$; by lemma 4.2, $v \in \mathrm{Reach}(x,S)$ so that $v$ is reachable from some node in $Y'$ through $S$. □

The next lemma follows directly from definition and from lemma 4.3

Lemma 4.4 $\mathrm{Nbrhd}(Y,S) = \mathrm{Nbrhd}(y,S)$,

$\mathrm{Reach}(Y,S) \cup Y = \mathrm{Reach}(y,S) \cup \{y\}$,

and $\mathrm{Closure}(Y,S) = \mathrm{Closure}(y,S)$. □

Lemma 4.5 For any subset $Y' \subset Y$, if $z \notin S \cup \text{Reach}(Y,S) \cup Y$ then $\text{Reach}(z,S \cup Y') = \text{Reach}(z,S)$.

Proof Since $z \notin \text{Reach}(Y,S)$, we have $Y \cap \text{Reach}(z,S) = \phi$. It then follows that $\text{Reach}(z,S) = \text{Reach}(z,S \cup Y')$. □

Lemma 4.6 For any subset $Y' \subset Y$, if $x \in Y \backslash Y'$, then $\text{Reach}(x,S \cup Y') \cup Y' = \text{Reach}(x,S)$.

Proof By lemma 4.2,

$$Y' \subset Y \subset \text{Reach}(y,S) \cup \{y\} = \text{Reach}(x,S) \cup \{x\}.$$

But $x \notin Y'$, so that $Y' \subset \text{Reach}(x,S)$. Furthermore if $v \in \text{Reach}(x,S \cup Y')$, by the definition of Y, the node v must be reachable from x through S. Thus

$$\text{Reach}(x,S \cup Y') \cup Y' \subset \text{Reach}(x,S).$$

The other inclusion is immediate. □

Corollary 4.7 If $x \in Y \backslash Y'$, then $|\text{Reach}(x,S \cup Y')| = |\text{Reach}(x,S)| - |Y'|$. □

Theorem 4.8 For any subset $Y' \subset Y$, if $x \in Y \backslash Y'$, $|\text{Reach}(x,S \cup Y')| \le |\text{Reach}(z,S \cup Y')|$ for all $z \notin S \cup Y'$.

Proof Let $x \in Y \backslash Y'$ and $z \notin S \cup Y'$. By theorem 4.1,

$$|\text{Reach}(x,S)| \le |\text{Reach}(z,S)|.$$

Then, by lemma 4.4 and corollary 4.6, $|\text{Reach}(x,S \cup Y')| \le |\text{Reach}(z,S \cup Y')|$. □

Theorem 4.8 has the following important implication: if y is the node selected in the minimum degree algorithm, the whole set Y can be numbered together.

## §4.2 Degree Update

Having selected and numbered the subset Y as defined in the previous section, we need to update the sizes of the new reachable sets in preparation for the next node selection step. For our discussion, we let S and Y be as before and define $\bar{S} = S \cup Y$. The problem is to determine $|\text{Reach}(u,\bar{S})|$ for all $u \in X \backslash S$.

Theorem 4.9 Let $u \in X\backslash\bar{S}$. Then

$$\text{Reach}(u,\bar{S}) = \begin{cases} (\text{Reach}(u,S) \cup \text{Reach}(Y,S))\backslash(Y \cup \{u\}), & \text{if } u \in \text{Reach}(Y,S) \\ \\ \text{Reach}(u,S) & \text{otherwise.} \end{cases}$$

Proof If $u \notin \text{Reach}(Y,S)$, it follows from lemma 4.4 that $\text{Reach}(u,\bar{S}) = \text{Reach}(u,S)$. On the other hand, when $u \in \text{Reach}(Y,S)$, the result follows from the definition of $\text{Reach}(u,\bar{S})$ and $\text{Reach}(Y,S)$. □

Theorem 4.9 says that only the nodes in Reach(Y,S) need to have their degrees updated. In addition, for $u \in \text{Reach}(Y,S)$, its new degree is given by

$$|\text{Reach}(u,\bar{S})| = |(\text{Reach}(u,S) \cup \text{Reach}(Y,S))\backslash(Y \cup \{u\})|.$$

Since the expression in theorem 4.7 can be written as the disjoint set union

$$[\text{Reach}(u,S)\backslash(Y \cup \text{Reach}(Y,S))] \cup [\text{Reach}(Y,S)\backslash\{u\}],$$

we have

$$|\text{Reach}(u,\bar{S})| = |\text{Reach}(u,S)\backslash(Y \cup \text{Reach}(Y,S))| + |\text{Reach}(Y,S)\backslash\{u\}|.$$

But the size of Reach(Y,S) is known, so the problem is reduced to the determination of

(4.3) $\text{Reach}(u,S)\backslash(Y \cup \text{Reach}(Y,S))$

for every $u \in \text{Reach}(Y,S)$.

As we do not intend to keep the set Reach(u S) but rather the number $|\text{Reach}(u,s)|$, the set given by (4.3) has to be generated in order to find its size. In what follows, we investigate the possibility of updating the degrees of a set of nodes in Reach(Y,S). Let $u \in \text{Reach}(Y,S)$. The following lemma is obvious.

Lemma 4.10 $\text{Nbrhd}(Y,S) \cup Y \in \mathcal{C}(\text{Nbrhd}(u,\bar{S}))$. □

Lemma 4.11 If $\text{Adj}(u) \subset \text{Nbrhd}(u,\bar{S}) \cup \text{Adj}(\text{Nbrhd}(u,\bar{S}))$, then

$|\mathcal{C}(\text{Nbrhd}(u,\bar{S}))| > 1$.

Proof Assume for contradiction that $|\mathcal{C}(\text{Nbrhd}(u,\bar{S}))| = 1$. By Lemma 4.10, $\text{Nbrhd}(Y,S) \cup Y = \text{Nbrhd}(u,\bar{S})$. But $\text{Adj}(u) \subset \text{Nbrhd}(u,\bar{S}) \cup \text{Adj}(\text{Nbrhd}(u,\bar{S}))$

$= \text{Nbrhd}(Y,S) \cup Y \cup \text{Adj}(\text{Nbrhd}(Y,S) \cup Y)$

$= \text{Nbrhd}(Y,S) \cup Y \cup \text{Reach}(Y,S)$

$= \text{Closure}\ (Y,S)$

$= \text{Closure}\ (y,S)$.

By the definition (4.2) of the set Y, the node u should belong to the set Y. □

Lemma 4.12 If $\text{Adj}(u) \subset \text{Nbrhd}(u,\bar{S}) \cup \text{Adj}(\text{Nbrhd}(u,\bar{S}))$, then

$\text{Adj}(\text{Nbrhd}(u,\bar{S})) = \text{Adj}(\text{Nbrhd}(u,\bar{S}) \cup \{u\}) \cup \{u\}$. □

Now, let $u \in \text{Reach}(Y,S)$ satisfy the conditions

(4.4) $\text{Adj}(u) \subset \text{Nbrhd}(u,\bar{S}) \cup \text{Adj}(\text{Nbrhd}(u,\bar{S}))$

and $|\mathcal{C}(\text{Nbrhd}(u,\bar{S}))| = 2$.

Then the following theorem holds.

Theorem 4.13 If $v \in \text{Reach}(Y,S)$ with

$\text{Adj}(v) \subset \text{Closure}(u,\bar{S})$

and $\text{Adj}(v) \cap (\text{Nbrhd}(u,\bar{S}) \setminus (\text{Nbrhd}(Y,S) \cup Y)) \neq \phi$,

then $|\text{Reach}(v,\bar{S})| = |\text{Reach}(u,\bar{S})|$.

Proof By lemma 3.3,

$$\mathrm{Nbrhd}(v,\bar{S}) \subset \mathrm{Nbrhd}(u,\bar{S}).$$

Together with $|\mathcal{C}(\mathrm{Nbrhd}(u,\bar{S}))| = 2$ and $\mathrm{Adj}(v) \cap (\mathrm{Nbrhd}(u,\bar{S}) \; (\mathrm{Nbrhd}(Y,S) \cup Y)) \neq \phi$, we have $\mathrm{Nbrhd}(v,\bar{S}) = \mathrm{Nbrhd}(u,\bar{S})$. Therefore, by lemma 4.12,

$$\begin{aligned} \mathrm{Reach}(v,\bar{S}) \cup \{v\} &= \mathrm{Adj}(\mathrm{Nbrhd}(v,\bar{S}) \cup \{v\}) \cup \{v\} \\ &\supset \mathrm{Adj}(\mathrm{Nbrhd}(v,\bar{S})) \cup \{v\} \\ &= \mathrm{Adj}(\mathrm{Nbrhd}(u,\bar{S})) \cup \{v\} \\ &= \mathrm{Reach}(u,\bar{S}) \cup \{u\}. \end{aligned}$$

By corollary 3.5, the two sets must be the same. □

Thus, when we update the degrees of the nodes in Reach(Y,S), it may be possible to update the degrees of a set of nodes $U \subset \mathrm{Reach}(Y,S)$, with only one application of the REACH operator.

## §4.3 Determination of Reachable Sets

The reach set operator

$$\mathrm{Reach}: X \backslash S \to P(X \backslash S)$$

is used repeatedly in our implementation of the minimum degree algorithm. In the minimum degree selection step, it is used to determine the set Y of nodes to be eliminated as given by (4.2). In the degree update step, the new degrees are obtained by finding the size of the set (4.3) through reachable sets. Thus, an efficient method for determining reachable sets is extremely important in the overall performance of the algorithm.

Let $y \in X \backslash S$. To find Reach(y,S) as $\mathrm{Adj}(\mathrm{Nbrhd}(y,S) \cup \{y\})$, it is apparent that all the neighbors of the nodes in $\mathrm{Nbrhd}(y,S) \cup \{y\}$ have to be inspected for membership. It is natural to ask whether it is possible to generate Reach(y,S) with less effort.

In this connection, we look for a subset V of S so that for $x \in X \backslash S$, the reachable set Reach(x,S) is the same as $\mathrm{Reach}(x,S \backslash V)$, where the latter operator is taken in the subgraph $G(X \backslash V) = (X \backslash V, E(X \backslash V))$. It is clear that this subset would satisfy the following conditions:

$$\mathrm{Nbrhd}(x, S \backslash V) \subset \mathrm{Nbrhd}(x, S)$$

and $$\mathrm{Reach}(x, S \backslash V) \backslash V = \mathrm{Reach}(x, S).$$

By marking the subset V as <u>inactive</u>, we need only to consider the subgraph $G(X \backslash V)$, and this can imply a drastic reduction in effort to generate $\mathrm{Reach}(x,S)$.

Consider the example in figure 4.1. The subset S contains 23 nodes marked with s. By making 7 of them inactive (those nodes marked in dark), the reachable sets are preserved in the subgraph. Furthermore, to find $\mathrm{Reach}(x,S)$, 16 nodes have to be inspected; whereas, in the subgraph, only 10 is necessary.

Consider an elimination step in the algorithm. Let S be the set of nodes already eliminated, and Y be the set (4.2) of nodes with minimum degree to be eliminated. If we use $\bar{S}$ to denote the new set of eliminated nodes $S \cup Y$, then

$$\mathrm{Nbrhd}(Y,S) \cup Y \in C(\bar{S}),$$

and $$\mathrm{Reach}(Y,S) = \mathrm{Adj}(\mathrm{Nbrhd}(Y,S) \cup Y).$$

All nodes in $\mathrm{Nbrhd}(Y,S) \cup Y$ except a subset R can be marked inactive so long as the subset R defines a <u>connected</u> subgraph G(R) and satisfies

(4.4) $\quad \mathrm{Reach}(Y,S) \subset \mathrm{Adj}(R).$

The best choice of R would be one with the smallest size satisfying the above conditions. For practical reasons, we look for one that can be determined easily.

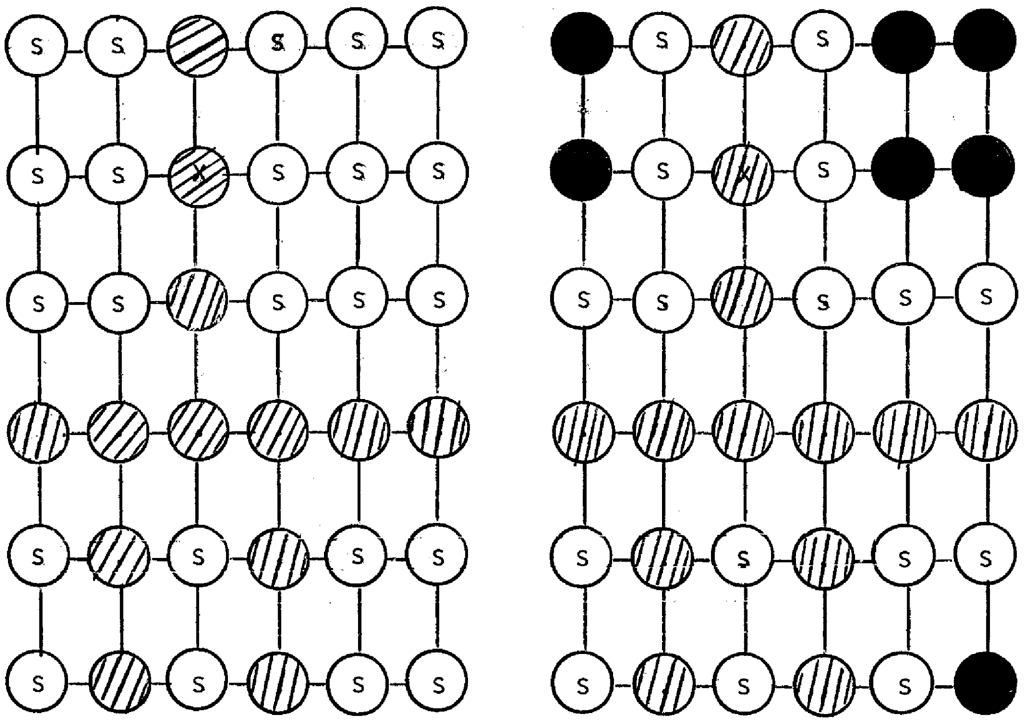


Figure 4.1 An example to illustrate the effect of de-activating eliminated nodes.

Obviously, the subset

(4.5) $\mathrm{Adj}(\mathrm{Reach}(Y,S)) \cap (\mathrm{Nbrhd}(Y,S) \cup S)$

satisfies (4.4), but it may not be connected. We simply take R to be a connected subgraph of $\mathrm{Nbrhd}(Y,S) \cup Y$ containing the set (4.5).

Thus, the way to "de-activate" eliminated nodes is completely specified if we can provide an algorithm for the following problem. We shall describe it in a general setting. Let $G= (X,E)$ be a connected graph and Z be a (possibly disconnected) subset of X. The problem is to determine a small subset $\bar{Z}$ such that

$$Z \subset \bar{Z} \subset X$$

and such that the subgraph $G(\bar{Z})$ is connected. The following algorithm serves the purpose though it may not produce a smallest $\bar{Z}$.

Step 1 (Initialization) Initialize $\bar{Z} \leftarrow Z$ and find a component $C \in \mathcal{C}(Z)$.

Step 2 (Test for termination) If $C = \bar{Z}$, stop.

Step 3 (Reach for another component) Determine a node $z \in Z \setminus C$, which is closest to some node c in C. Let $c, x_1, \ldots, x_t, z$ be a shortest path with $x_i \in X \setminus Z$, $1 \le i \le t$.

Step 4 (Expansion of component) Find the component $C' \in \mathcal{C}(Z)$ with $z \in C'$. Put

$$\bar{Z} \leftarrow \bar{Z} \cup \{x_1, \ldots, x_t\}$$

and $C \leftarrow C \cup \{x_1, \ldots, x_t\} \cup C'$.

Step 5 (Loop) Go to step 2.

## §4.4 An Overview of the Refined Algorithm

Refinements have been given to improve the overall performance of the minimum degree algorithm as described in section 3.2. In this section, we combine them and provide an overall picture of the entire algorithm.

Step 1 (Initialization) Let $G = (X,E)$ and initialize $S \leftarrow \phi$, $DEG(x) \leftarrow |Adj(x)|$, for $x \in X$.

Step 2 (Minimum degree selection) Pick a node $y \in X \setminus S$ where $DEG(y) = \min_{x \in X \setminus S} DEG(x)$.

**Step 3 (Mass minimum degree elimination) Determine the sets Reach(y,S)** and Closure(y,S). Then find $Y = \{x \in Reach(y,S) \cup \{y\} \mid Adj(x) \subset Closure(y,S)\}$, and number the nodes in Y next.

Step 4 (Mass degree update) For $u \in Reach(Y,S)$, determine $Reach(u,S) \setminus (Reach(Y,S) \cup Y)$ and put $DEG(u) \leftarrow |Reach(u,S) \setminus (Reach(Y,S) \cup Y)| + |Reach(Y,S) \setminus \{u\}|$. Check for mass degree update conditions and perform the update accordingly, if possible.

Step 5 (Deactivate some eliminated nodes) Find a subset V of $Nbrhd(Y,S) \cup Y$ such that

$G((Nbrhd(Y,S) \cup Y) \setminus V)$ is connected

and $Reach(Y,S) \subset Adj((Nbrhd(Y,S) \cup Y) \setminus V)$.

Then put

$(X,E) \leftarrow (X \setminus V, E(X \setminus V))$

and $S \leftarrow (S \cup Y) \setminus V$.

**Step 6 (Loop or stop) If $S = X$, stop. Otherwise, go to step 2.**

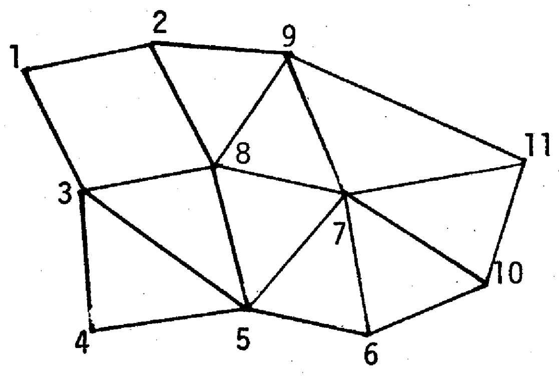
In the next sections, we shall consider some implementation details and provide some experimental results on the performance of this scheme.

## §5 Implementation Details and Numerical Experiments

In this section we describe the important aspects of our implementations (MD), and report some numerical experiments showing its performance. As a basis for comparison, we have used the appropriate routines from the Yale Sparse Matrix Package (YSMP) [9], which is fairly widely distributed and enjoys a good reputation.

In order to avoid doing any searching for a node of minimum degree, the degrees of the nodes are stored using the three arrays ORG, NEXT, and PREV, each of length N, as depicted in Figure 5.1. Nodes having the same degree are stored in a doubly linked degree list; the beginning of the list containing nodes of degree i is stored in ORG(i), and the arrays NEXT and PREV contain the usual forward and backward pointers of a doubly linked list. If node q is the first in the degree list, then PREV(q) is -deg(q), and if node q is the last node in the degree list, then NEXT(q) = 0. Note that when the degree of a node is changed, it can be deleted from the old degree list and inserted in the new list in a fixed number of operations, independent of N. By maintaining a pointer to the first nonzero entry in ORG, we can easily find a node of minimum degree [6].

In the set of subroutines which implement our algorithm, the set S is maintained using a three-state array SMASK, where SMASK(i) > 0 <=> node i $\in$ S, SMASK(i) = 0 <=> node i $\notin$ S, and SMASK(i) < 0 if node i has been deactivated using the algorithm of section 4.3. In addition, four other arrays of length N are used, one of which is, like SMASK, only required to store these different states of a node. The graph itself is stored as a sequence of adjacency lists in the array pair (XADJ, ADJNCY), as shown in Figure 5.1. Since our implementation does not exploit the fact that two of



Graph G

| | ORG | NEXT | PREV |
|---|---|---|---|
| 1 | 0 | 4 | -2 |
| 2 | 1 | 11 | 6 |
| 3 | 6 | 9 | -4 |
| 4 | 3 | 0 | 1 |
| 5 | 8 | 0 | 8 |
| 6 | 7 | 2 | -3 |
| 7 | 0 | 0 | -6 |
| 8 | 0 | 5 | -5 |
| 9 | 0 | 0 | 3 |
| 10 | 0 | 0 | 11 |
| 11 | 0 | 10 | 2 |

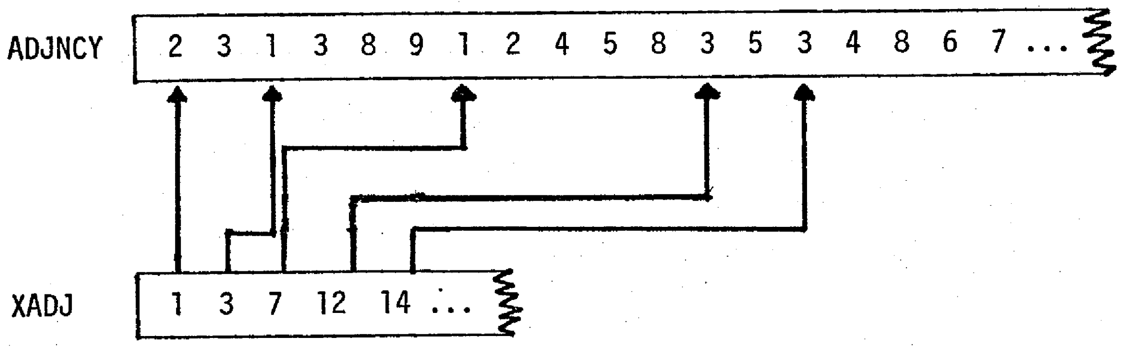


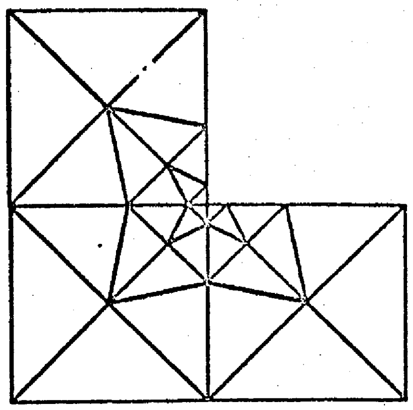
Figure 5.1 Representation of the adjacency and degree structure of the graph using the arrays XADJ, ADJNCY, ORG, NEXT, and PREV.

the vectors of length need only represent three states, our total array storage requirements for the ordering program is $9N + 2|E|$ integer locations.

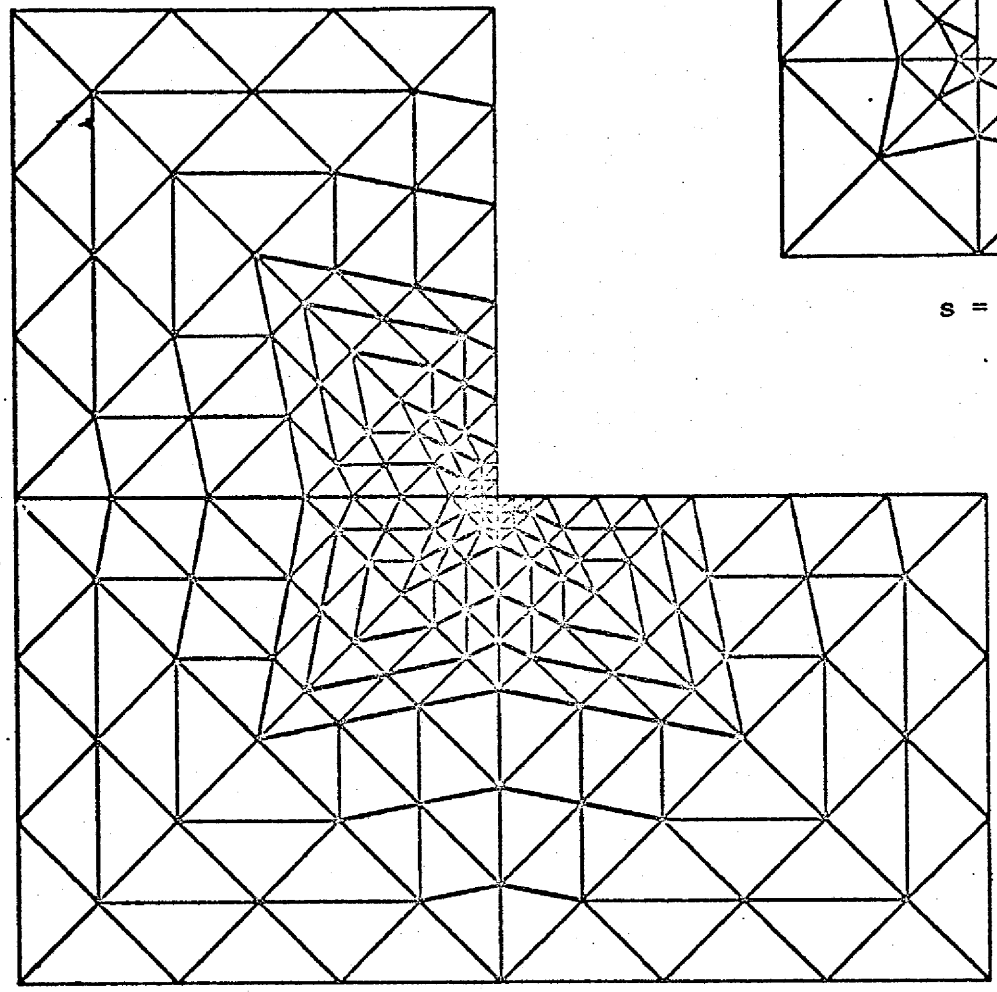
In order to obtain some evidence bearing on the asymptotic execution time of our implementation, the program was applied to the graded L mesh (graph) shown in Figure 5.2, subdivided by increasing subdivision factors s, yielding $s^2$ as many triangles as in the original mesh. The YSMP subroutine SORDER was also run on the same problems to obtain comparative results. The results are summarized in Table 5.1. Execution times are in seconds on an IBM 360/75 computer. The programs are written in Fortran, compiled using the optimizing version of the H-level compiler.

The entries in the storage column for the YSMP program were obtained by inserting a statement in SORDER to monitor the maximum storage used in some working storage arrays. The user must estimate this number, and provide at least this much storage to allow the program to execute. This is a disadvantage shared by many minimum degree algorithm implementations. **We regard the fact that our implementation uses a modest fixed amount of storage as a major advantage. The <u>quality</u> of the orderings produced by the** two programs was almost identical; i.e., the amount of fill suffered by the corresponding ordered matrices when factored was nearly identical.

Note that the asymptotic execution time of MD appears to be of a lower order than the YSMP program. Indeed, the numerical results in Table 5.1 suggest quite strongly that the execution time of our program is O(N). Other similar experiments with finite element discretizations of two and three dimensional regions suggest that our program runs in O(N) time for this class of problems.



s = 1



s = 3

Figure 5.2  Graded L mesh

|  | EXECUTION TIME |  |  |  | STORAGE |  |
|---|---|---|---|---|---|---|
|  | MD |  | YSMP |  |  |  |
| N | TIME | $\frac{\text{TIME}}{\text{N}}$ | TIME | $\frac{\text{TIME}}{\text{N}}$ | MD | YSMP |
| 265 | 1.20 | 4.52 | .90 | 3.21 | 4139 | 6667 |
| 406 | 1.97 | 4.85 | 1.50 | 3.65 | 6371 | 10428 |
| 577 | 2.95 | 5.11 | 2.31 | 3.85 | 9083 | 15559 |
| 778 | 3.89 | 5.00 | 3.39 | 4.27 | 12275 | 23044 |
| 1009 | 5.14 | 5.10 | 4.60 | 4.45 | 15947 | 30211 |
| 1270 | 6.30 | 4.96 | 6.28 | 4.95 | 20099 | 40220 |
| 1561 | 7.98 | 5.11 | 8.14 | 5.21 | 24731 | 49407 |
| 1882 | 9.87 | 5.24 | 10.65 | 5.66 | 29843 | 61488 |
| 2233 | 11.98 | 5.37 | 12.68 | 5.68 | 35435 | 76427 |
| 2614 | 13.63 | 5.21 | - | - | 41507 | - |
| 3025 | 15.99 | 5.29 | - | - | 48059 | - |
| 3466 | 18.50 | 5.34 | - | - | 55091 | - |
|  |  | $(\times 10^{-3})$ |  | $(\times 10^{-3})$ |  |  |

Table 5.1 Comparisons of execution times and storage requirements for the MD and YSMP programs.

In order to demonstrate the effectiveness of (a) the deactivation technique described in section 4.3, and (b) the "mass degree update" technique (Theorem 4.13), we solved the problems reported on above, with one or both of these features removed from our program. The results are summarized in Table 5.2, and illustrate their value for large N.

| N | NODE DEACTIVATION AND MASS DEGREE UPDATE | NO NODE DEACTIVATION BUT MASS DEGREE UPDATE | NODE DEACTIVATION BUT NO MASS DEGREE UPDATE | NO NODE DEACTIVATION AND NO MASS DEGREE UPDATE |
|---|---|---|---|---|
| 265 | 1.20 | 1.30 | 1.49 | 1.73 |
| 406 | 1.97 | 2.16 | 2.81 | 3.61 |
| 577 | 2.95 | 3.28 | 4.15 | 5.82 |
| 778 | 3.89 | 4.30 | 5.90 | 8.98 |
| 1009 | 5.14 | 5.82 | 7.87 | 13.57 |
| 1270 | 6.30 | 7.07 | 10.65 | 18.24 |
| 1561 | 7.98 | 9.04 | 13.92 | 24.98 |
| 1882 | 9.87 | 11.78 | 17.66 | 32.39 |
| 2233 | 11.98 | 14.49 | 21.74 | 41.81 |

Table 5.2 Results showing the effect of node deactivation and mass degree update.

## §6 Conclusions

We have described an implementation of the minimum degree algorithm which requires only O(N) storage in addition to that required for the original graph. The storage required by the program is known before execution, and is independent of the amount of fill that is suffered by the correspondingly reordered matrix, when it is factored. The results of Table 5.2 show that the success of our approach depends upon two important techniques; "mass degree update", and "node deactivation". The latter of these techniques has important application in other areas of sparse matrix computation, particularly in the area of storage allocation and related problems [4]. Finally, the results in Table 5.1, along with other experiments, suggest that the execution time of our implementation, for problems arising in finite element applications is O(N). Unfortunately, it is not difficult to construct mesh-like problems where the running time of our algorithm is superlinear.

## §7 References

[1] I.S. Duff, A.M. Erisman and J.K. Reid, "On George's nested dissection algorithm", SIAM J. Numer. Anal., 5 (1976), pp. 686-695.

[2] 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).

[3] Alan George and David R. McIntyre, "On the application of the minimum degree algorithm to finite element systems", SIAM J. Numer. Anal., to appear.

[4] Alan George and Joseph W.H. Liu, "On finding diagonal block envelopes of triangular factors of partitioned matrices", Dept. of Computer Science Technical Report, CS-77-10, 1977.

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

[6] J.K. Reid, Private communication.

[7] 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).

[8] D. Rose, R. Tarjan and G. Luecker, "Algorithmic aspects of vertex elimination on directed graphs", SIAM. J. Computing, 5 (1975), pp. 266-283.

[9] Andrew H. Sherman, "Yale Sparse Matrix Package Users Guide", Lawrence Livermore Laboratory Report, UCID-30114, August 1975.