Printing Requisition / Graphic Services

45610

1. **Please complete unshaded areas** on form as applicable.
2. **Distribute copies** as follows: White and Yellow to Graphic Services. **Retain Pink Copies for your records.**
3. On completion of order the **Yellow copy will be returned** with the printed material.
4. **Please direct enquiries, quoting requisition number and account number,** to **extension 3451.**

TITLE OR DESCRIPTION: CS-78-11

DATE REQUISITIONED: Oct. 23 | DATE REQUIRED: ASAP | ACCOUNT NO.: 1,2,6,4,4,3,1,0,2

REQUISITIONER – PRINT: Sue DeAngelis | PHONE: 2172 | SIGNING AUTHORITY: Sue DeAngelis

MAILING INFO – NAME | DEPT. | BLDG. & ROOM NO. | ☐ DELIVER ☒ PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 28 | NUMBER OF COPIES: 1

TYPE OF PAPER STOCK: ☒ BOND ☐ NCR ____ PT. ☐ COVER ☐ BRISTOL ☐ SUPPLIED ☐ ____

PAPER SIZE: ☒ 8½ x 11 ☐ 8½ x 14 ☐ 11 x 17 ☐ ____

PAPER COLOUR: ☒ WHITE ☐ ____ | INK: ☒ BLACK ☐ ____

PRINTING: ☐ 1 SIDE ____ PGS. ☒ 2 SIDES ____ PGS. | NUMBERING: FROM ____ TO ____

BINDING/FINISHING: ☒ COLLATING ☐ STAPLING ☐ ____ HOLE PUNCHED ☐ PLASTIC RING

FOLDING/PADDING | CUTTING SIZE

Special Instructions

COPY CENTRE — OPER. NO. | BLDG. | MACH. NO.

DESIGN & PASTE-UP

OPER. NO.	TIME	LABOUR CODE
		D01
		D01
		D01

TYPESETTING

	QUANTITY	OPER. NO.	TIME	LABOUR CODE
PAP000000				T01
PAP000000				T01
PAP000000				T01

PROOF

PRF				
PRF				
PRF				

NEGATIVES

	QUANTITY	OPER. NO.	TIME	LABOUR CODE
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01

PMT

PMT				C01
PMT				C01
PMT				C01

PLATES

PLT				P01
PLT				P01
PLT				P01

STOCK

				001
				001
				001
				001

BINDERY

RNG				B01
RNG				B01
RNG				B01
MIS000000				B01

OUTSIDE SERVICES

$ ____ COST

TAXES – PROVINCIAL ☐ FEDERAL ☐

GRAPHIC SERV. OCT. 85 482-2

An Optimal Algorithm for Symbolic Factorization of Symmetric Matrices*

by

Alan George[†]
and
Joseph W.H. Liu[††]

Research Report CS 78-11

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

[††]Systems Dimensions Ltd
111 Avenue Road
Toronto, Ontario, Canada

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

Abstract

A fundamental problem in the computer solution of a sparse, N by N, positive definite system of equations Ax=b is, given the structure of A, to determine the structure of its Cholesky factor L, where $A = LL^T$. This problem arises because it is often desirable to set up a data structure for L before the numerical computation is performed, and in order to do this we must know the positions of the nonzeros of L. We describe a representation $\mathcal{R}_L$ for L which typically requires far fewer data items than the number of nonzeros in L, and an algorithm is then described which generates $\mathcal{R}_L$. The time and space complexity of the algorithm is shown to be $O(|A|, |\mathcal{R}_L|)$, and can never be worse than $O(|L|)$. Here $|\mathcal{R}_L|$ denotes the number of items in the data structure for L, and |A| and |L| denote the number of nonzeros in A and L respectively. For a certain class of problems, we show that the execution time of the algorithm is O(N), even though |L| is O(N log N). We also provide some numerical results showing that the algorithm can be implemented so that the program performance reflects its theoretically predicted behaviour.

1. Introduction

Consider the symmetric positive definite system of linear equations

(1.1) $Ax=b$,

where A is N by N and sparse. If we solve (1.1) using Cholesky's method, the system is usually first reordered by an algorithm such as the minimum degree algorithm [4], so that its Cholesky factor suffers low fill-in. We do not deal with this reordering problem here, and simply assume that (1.1) has already been reordered appropriately.

Since L is sparse, the next step in the solution process is to construct a data structure for L, so that only its nonzeros are stored. In order to do this we must know the positions of the nonzeros of L, leading to the problem which we deal with in this paper. That is, given the structure of A, we want to determine the corresponding structure of its Cholesky factorization. Since the process is entirely logical, involving no numerical computation, it is often referred to as "symbolic factorization".

At least two such algorithms for computing the fill-in (or the structure of L) have been described in the literature [8,9]. The algorithms are quite similar, and their time complexity has been shown to be $O(|L|)$, where $|L|$ denotes the number of nonzeros in L. If the desired output of the algorithms is the positions of the nonzeros in L, then the immediate implication is that these algorithms are asymptotically optimal.

However, in practice, the reason for performing symbolic factorization is to construct a storage scheme for L, and very often these data structures involve far fewer than $|L|$ items of information, [9].

For example, for one such data structure and problem class, the number of data structure pointers etc. is O(N), even though |L| is O(N log N) [3]. In this context it seems reasonable to regard an algorithm as optimal only if its execution time and its space requirement are $O(|\mathcal{R}_L|, |A|)$, where $\mathcal{R}_L$ is the representation used to describe the structure of L.

In this paper we describe an algorithm for symbolic factorization which generates a specific representation for L. The time complexity of the algorithm is shown to be optimal in the above sense (i.e., $O(|\mathcal{R}_L|, |A|)$) and can never be worse than O(|L|). For a special class of finite element problems, appropriately ordered, we show that the execution time of the algorithm is O(N), even though |L| is **O(N log N).**

In general, we note that the representation produced may not be optimal with respect to $|\mathcal{R}_L|$. However, the algorithm is optimal for producing this representation.

Our algorithm has another important feature not present in previous algorithms. The algorithm <u>does not use any of its output during execution</u>; it operates only on the graph of A. Thus, if insufficient space is available to retain $\mathcal{R}_L$, it can be discarded; the algorithm can still execute and determine the <u>number</u> $|\mathcal{R}_L|$, thus furnishing the amount of space needed for $\mathcal{R}_L$. Alternatively, $\mathcal{R}_L$ can be written on secondary storage as it is generated. In either case, we **argue that our algorithm is optimal in terms of space requirements.**

An outline of our paper is as follows. In section 2 we describe a useful representation R_L for the structure of L which is the basis for at least two efficient storage schemes for sparse matrices [3, 9]. Section 3 contains a review of a quotient graph model of symmetric factorization, developed by the authors in [5]. Section 4 contains a development of the symbolic factorization algorithm, using the quotient graph model, and section 5 contains an example illustrating that its execution can be of an order lower than $|L|$. Section 6 contains a few numerical experiments showing that the algorithm can be implemented efficiently, along with a discussion of its advantages and disadvantages compared to alternative symbolic factorization algorithms.

§2 Representation of the Structure of L

In this section we consider the problem of efficiently representing the structure of the lower triangular matrix L. This task is clearly fundamental in the design of an efficient data structure for storing L.

It is often true that when symmetric Gaussian elimination is applied to matrices which have been ordered so as to suffer low fill-in, the resulting factor L has many "similar" columns. That is, for some column k, there are several columns j, $j > k$, having essentially the same nonzero structure. This fact motivates the following definition.

Given an N by N lower triangular matrix L, column k is a representative for column j, $j > k$, if and only if for all $i \geq j$, $L_{ij} \neq 0 \iff L_{ik} \neq 0$. Simply stated, column k represents column j if they have identical structure below position j. Note that column k represents itself.

If we know the structure of column k, and we know which columns it represents, then their structure is completely determined. Since we are interested in representing the entire structure of L, we are led to the following definitions.

A representative map M for L is an ordered set of integers $(m_1, m_2, \ldots, m_N)$ such that $1 \leq m_j \leq j$, and such that $m_j = k$ implies that column k is a representative for column j. The subset $M^* \subset M$ consisting of distinct members of M is called a representative set for L. Finally, a representation for L by M is the set $\mathcal{R}_L$, where $\mathcal{R}_L$ is given by

$$\mathcal{R}_L = \{(i,j) \mid j \in M^*, L_{ij} \neq 0\}$$

An example illustrating these definitions is given in Figure 2.1.

$$L = \begin{pmatrix}
X & & & & & & & & & \\
 & X & & & & & & & & \\
 & X & X & & & & & & & \\
 & & & X & & & & & & \\
 & & X & X & X & & & & & \\
 & & & & & X & & & & \\
X & & X & X & X & & X & & & \\
X & & X & X & X & & X & X & & \\
 & & & X & X & & & & X & \\
 & & & & & X & & X & & X \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{pmatrix}$$

$M = (1,2,3,4,4,6,1,6,4,6)$

$M^* = \{1,2,3,4,6\}$

$R_L = \{(7,1), (8,1), (3,2), (5,3), (7,3), (8,3), (5,4), (7,4), (8,4), (9,4), (10,6)\}$

Figure 2.1 An example illustrating the sets M, M^* and R_L .

A representative map is said to be <u>monotone</u> if $m_i \leq m_{i+1}$, $1 \leq i < N$. In this case it is clear that M^* and R_L are sufficient to describe the structure of L, since M can be inferred from M^*.

§3 The Quotient Graph Model of Symmetric Elimination

3.1 Graph Theoretic Preliminaries

Symbolic factorization is the process of simulating the numerical factorization of a given matrix A in order to obtain the zero-nonzero structure of its factor L. Since the numerical values of the matrix components are of no significance in this connection, the problem can be conveniently studied using a graph theory model. The readers are assumed to know the basic notions in graph theory, reference to which can be found in [1]. We now introduce some more definitions and establish results that are pertinent in the study of the elimination process.

Let $G = (X,E)$ be a given undirected graph. Consider a subset $S \subset X$ and a node $y \in X-S$. The node y is said to be reachable from a node x through S if there exists a path $(x, s_1,\ldots,s_t,y)$ such that $s_i \in S$ for $1 \leq i \leq t$. The reachable set of y through S is then defined to be

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

A related notion is the neighbourhood set of y in S, which is the subset

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

For convenience in later discussions, we define the closure of y in S to be

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

We now review the relevance of these notions in 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 the one for which X^A is labelled from 1 to N:

$$X^A = \{x_1, x_2, \ldots, x_N\},$$

and $\{x_i, x_j\} \in E^A$ if and only if $A_{ij} \neq 0$

Let $A = LL^T$, where L is the Cholesky factor of A. The matrix $F = L + L^T$ is called the filled matrix of A and the corresponding graph $G^F = (X^A, E^F)$ the filled graph of G^A. The following result relates E^F directly to E^A using the reachable set notion. This idea was discovered independently in [8]. Let $S_i = \{x_1, \ldots, x_i\}$ for $1 \leq i \leq N$ and $S_0 = \phi$.

Lemma 3.1 [4] Let $j > i$. The unordered pair $\{x_i, x_j\} \in E^F$ if and only if $x_j \in \text{Reach}\,(x_i, S_{i-1})$. □

Corollary 3.2 [4] $|E^F| = \sum_{i=1}^{N} |\,\text{Reach}\,(x_i, S_{i-1})\,|$. □

In subsequent sections of this paper, these various graph theoretic notions will be applied to different graphs. When the graph is not clear from context, we will attach the appropriate subscript to the nomenclature. Thus, notations of the following type will be used: $\text{Adj}_G(y)$, $\deg_G(y)$, $\text{Reach}_G(y,S)$, etc.

3.2 The Quotient Graph Model

In [5], the authors introduced a quotient graph model for the study of the Gaussian elimination process. We now briefly review this model.

The central notion is that of a quotient graph. Let $G = (X,E)$ be a given graph. Let $\mathcal{P}$ be a partitioning of the node set X:

$$\mathcal{P} = \{Y_1, Y_2, \ldots, Y_p\}.$$

That is, $\bigcup_{k=1}^{p} Y_k = X$ and $Y_i \cap Y_j = \phi$ for $i \neq j$.

The quotient graph of G with respect to $\mathcal{P}$ is defined to be the graph $(\mathcal{P},E)$, where $\{Y_i, Y_j\} \in E$ if and only if $Y_i \cap \text{Adj}\ (Y_j) \neq 0$. This graph will be denoted by $G/\mathcal{P}$.

An important type of partitioning is that defined by connected components. Let S be a subset of X. The component partitioning $C(S)$ of S is defined as

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

This can be extended to a partitioning on X. We define the partitioning on X induced by S to be

$$\bar{C}(S) = C(S) \cup \{\{x\} \mid x \in X\text{-}S\}.$$

We are now ready to model the elimination process as a sequence of quotient graphs. Let $G^A = (X^A, E^A)$ be the graph and $x_1, x_2, \ldots, x_N$ be the sequence of node elmination. As in section 3.1, let $S_i = \{x_1, \ldots, x_i\}$ for $1 \leq i \leq N$ and $S_0 = \phi$.

Consider the partitioning $\bar{C}(S_i)$ induced by S_i and the corresponding quotient graph $G/\bar{C}(S_i)$. We shall denote this quotient graph by

G_i. In this way, we obtain a sequence of quotient graphs

$$G_1, G_2, \ldots, G_N.$$

We quote the following result from [5].

Theorem 3.3 For $y \notin S_i$,

$$\text{Reach}_{G_i}(\{y\}, C(S_i)) = \{\{x\} \mid x \in \text{Reach}_G A(y, S_i)\}. \quad \square$$

By lemma 3.1, the filled graph G^F is characterised by the set of reachable sets $\text{Reach}_G A(x_i, S_{i-1})$. Then, the above theorem implies that the filled graph can be implicitly represented by the sequence of quotient graphs.

The primary advantage of the quotient graph model is that its computer implementation is very efficient. In particular, it can be implemented in space proportional to the number $|E^A|$ of nonzeros in A. The in-place implementation of the model relies on the following results quoted from [5].

Theorem 3.4 Let $G_i = (\bar{C}(S_i), E_i)$, $1 \le i \le N$ be the sequence of quotient graphs. Then, for $1 \le i \le N$

$$|\bar{C}(S_{i+1})| \le |\bar{C}(S_i)|,$$

and $$|E_{i+1}| \le |E_i| \quad \square$$

Theorem 3.5 For $x \notin S_{i+1}$

$$|\text{Adj}_{G_{i+1}}(\{x\})| \le |\text{Adj}_{G_i}(\{x\})| \quad \square$$

For details of the in-place implementation, the reader is referred to [5]. In the next subsection, we shall consider some properties of the induced partitionings $\bar{C}(S_i)$.

3.3 The Component Partitionings $C(S_i)$

In the formulation of the quotient graph model, the sequence of node subsets

$$S_1, S_2, \ldots, S_N$$

defines a sequence of component partitionings

$$C(S_1), C(S_2), \ldots, C(S_N).$$

We first establish the relation between $C(S_{i-1})$ and $C(S_i)$ through a series of lemmas.

Lemma 3.6 Nbrhd $(x_i, S_{i-1}) = \cup \{Y \in C(S_{i-1}) \mid x_i \in \text{Adj}(Y)\}$.

Proof Consider any $Y \in C(S_{i-1})$ with $x_i \in \text{Adj}(Y)$. For $y \in Y \subset S_{i-1}$, since $x_i \in \text{Adj}(Y)$, y is reachable from a subset of Y and hence $y \in$ Nbrhd (x_i, S_{i-1}).

On the other hand, consider $y \in$ Nbrhd (x_i, S_{i-1}). Let $(x_i, s_1, \ldots, s_t, y)$ be a path where $\{s_1, \ldots, s_t, y\} \subset S_{i-1}$. Define $Y \in C(S_{i-1})$ such that $\{s_1, \ldots, s_t, y\} \subset Y$. Clearly Adj (Y) contains x_i and hence the result. □

Lemma 3.7 $\{x_i\} \cup$ Nbrhd $(x_i, S_{i-1}) \in C(S_i)$.

Proof Clearly $\{x_i\} \cup$ Nbrhd $(x_i, S_{i-1}) \subset S_i$ and is connected. That it is a maximal connected set in $G(S_i)$ is left as an exercise. □

Theorem 3.8 $C(S_i) = \{\{x_i\} \cup \text{Nbrhd}(x_i, S_{i-1})\} \cup$

$$\{Y \in C(S_{i-1}) \mid x_i \notin \text{Adj}(Y)\}.$$

Proof It follows from lemmas 3.6 and 3.7. □

We now establish an interesting property on the component partitionings $C(S_i)$. Define the set

$$X = \cup \{C(S_i) \mid 1 \leq i \leq N\},$$

that is, X contains all the component subsets in all $C(S_i)$.

Theorem 3.9 $X = \{\{x_i\} \cup \text{Nbrhd}\ (x_i, S_{i-1}) \mid 1 \le i \le N\}$

Proof By lemma 3.7, it suffices to show that

$$C(S_k) \subset \{\{x_i\} \cup \text{Nbrhd}\ (x_i, S_{i-1}) \mid 1 \le i \le N\}.$$

for k=1,...,N. But this can be proved by induction on k using theorem 3.8. □

There is therefore a one-to-one correspondence between the node set X and the component set X. Indeed, the mapping is given by

$$x_i \Longleftrightarrow \{x_i\} \cup \text{Nbrhd}\ (x_i, S_{i-1}).$$

This correspondence will be useful in establishing complexity bounds for the symbolic factorization algorithm. The following result relates the set X with $\text{Reach}\ (x_i, S_{i-1})$.

Lemma 3.10 $\text{Reach}\ (x_i, S_{i-1}) = \text{Adj}\ (\{x_i\} \cup \text{Nbrhd}\ (x_i, S_{i-1}))$.

□

§4 Symbolic Factorization Using the Quotient Graph Model

4.1 Symbolic Factorization

Consider a symmetric matrix A with Cholesky factor L. Let $G^A = (X^A, E^A)$ be its associated graph, where

$$X^A = \{x_1, x_2, \ldots, x_N\}.$$

In view of the result of lemma 3.1, symbolic factorization of A may be regarded as the determination of the sets

$$\text{Reach}_{G^A}(x_i, \{x_1, \ldots, x_{i-1}\})$$

for i=1,...,N.

These reachable sets in the graph G^A can, however, be determined in terms of the structure in the quotient graphs. The correspondence is given in theorem 3.3. With this connection, we can state a symbolic factorization algorithm in terms of quotient graphs.

Step 0 (Initialization) Let $S_0 = \phi$, $G_0 = G^A/\bar{C}(S_0)$.

Also put $i \leftarrow 1$.

Step 1 (Reachable set determination) Find the reach set

$$\text{Reach}_{G_{i-1}}(\{x_i\}, C(S_{i-1}))$$

in the quotient graph G_{i-1}.

Step 2 (Quotient graph transformation) Form $S_i \leftarrow S_{i-1} \cup \{x_i\}$ and $C(S_i)$ as given by theorem 3.8.

Perform the in-place transformation [5] of the quotient graph G_i from G_{i-1}, where

$$G_i = G^A/\bar{C}(S_i).$$

Step 3 (Loop or stop) Set $i \leftarrow i + 1$. If $i > N$, stop; otherwise go to step 1.

To consider the complexity of the algorithm, we first study the overall contribution from step 1. The next lemma shows that it is bounded by $O(|E^F|)$. In the next section, we shall show that step 2 can be implemented with the same bound. Thus, it is an $O(|E^F|)$ algorithm.

Lemma 4.1 The overall complexity in the reachable set determination step is $O(|E^F|)$.

Proof Consider the i-th step in determining

$$\text{Reach}_{G_{i-1}}(\{x_i\}, C(S_{i-1})).$$

This can be done by inspecting

$$\text{Adj}_{G_{i-1}}(\{x_i\})$$

and $$\text{Adj}_{G_{i-1}}(Y), \text{ for every } Y \in C(S_{i-1}) \cap \text{Adj}_{G_{i-1}}(\{x_i\}) .$$

By theorem 3.5,

$$\sum_i | \text{Adj}_{G_{i-1}}(\{x_i\}) | \leq \sum_i | \text{Adj}_G(x_i) | = O(|E^A|).$$

On the other hand, by theorem 3.9 and lemma 3.10,

$$\sum_i \{|\text{Adj}_{G_{i-1}}(Y)| \mid Y \in C(S_{i-1}) \cap \text{Adj}_{G_{i-1}}(\{x_i\})\}$$

$$= \Sigma \{|\text{Adj}(Y)| \mid Y \in X\}$$

$$= \Sigma \{|\text{Reach}(x_i, S_{i-1})| \mid 1 \leq i \leq N\}$$

$$= O(|E^F|).$$

Combining, we have shown that the complexity is $O(|E^F|)$. □

4.2 Incomplete Quotient Graph Transformation

In this section, we shall introduce the technique of incomplete tranformation, which helps to reduce the amount of work in step 2 of symbolic factorization as described in section 4.1. This basic idea is similar to that exploited by Rose et.al. [8] and Sherman [9], although here the technique is presented in the context of quotient graph transformations. We first prove a lemma. Recall that

$$X = \cup \{C(S_i) \mid 1 \leq i \leq N\}.$$

Lemma 4.2 Let $Y \in X$. Then $Y \in \text{Adj}_{G_{i-1}}(\{x_i\})$ if and only if

$$\min \{k \mid x_k \in \text{Adj}_G(Y)\} = i.$$

Proof "if part" Assume $\min \{k \mid x_k \in \text{Adj}_G(Y)\} = i$.

By theorem 3.9, let

$$Y = \{x_j\} \cup \text{Nbrhd}\ (x_j, S_{j-1})$$

Since the minimum subscript of neighbours of Y is i, by theorem 3.8, we have

$$Y \in C(S_j) \cap C(S_{j+1}) \ldots \cap C(S_{i-1}).$$

In other words, $Y \in \bar{C}(S_{i-1})$ so that $Y \in \text{Adj}_{G_{i-1}}(\{x_i\})$.

"only if part" Assume $Y \in \text{Adj}_{G_{i-1}}(\{x_i\})$.

Clearly $x_i \in \text{Adj}_G(Y)$. If $Y = \{x_j\} \cup \text{Nbrhd}\ (x_j, S_{j-1})$, we have $Y \in C(S_j) \cap \ldots \cap C(S_{i-1})$. That means $\text{Adj}(Y) \cap \{x_{j+1}, \ldots, x_{i-1}\} = \phi$, and hence the result. □

The idea of incomplete quotient graph transformation is based on Lemma 4.2. In performing the transformation

$$G_{i-1} = (\bar{C}(S_{i-1}),\ E_{i-1}) \rightarrow G_i = (\bar{C}(S_i),\ E_i),$$

if $Y = \{x_i\} \cup \text{Nbrhd}\ (x_i, S_{i-1})$, the complete transformation requires the setting up of

$$\text{Adj}_{G_i}(Y) = \{\{x\} \mid x \in \text{Adj}_G(Y)\}$$

and for $x \in \text{Adj}_G(Y)$,

$$\text{Adj}_{G_i}(\{x\}) = (\text{Adj}_{G_{i-1}}(\{x\}) \cup \{Y\}) \cap \bar{C}(S_i).$$

However, in view of lemma 4.2, we do not have to form all the $\text{Adj}_{G_i}(\{x\})$ for every $x \in \text{Adj}_G(Y)$. Instead, the neighbours update needs only be performed on the $x \in \text{Adj}_G(Y)$ with the smallest subscript. Thus, we have proved the following result.

Lemma 4.3 The overall complexity in the incomplete quotient graph step is $O(|E^F|)$. □

4.3 Mass Symbolic Elimination

The results of lemmas 4.1 and 4.3 imply that the symbolic factorization process can be implemented in time proportional to $O(|E^F|)$. In this section, we discuss another enhancement so that the time complexity becomes $O(|R_L|)$, where R_L is the representation used to describe the structure of L, discussed in section 2.

The idea for the enhancement is motivated from an implementation of the minimum degree algorithm by the authors [4]. We first quote some results about reachable sets.

Let $G = (X,E)$. Consider a subset $S \subset X$, and a node $y \notin S$.

Lemma 4.4 [4] Let $x \in X\text{-}S$. If

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

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

Corollary 4.5 Let x be as in lemma 4.4. Then

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

Proof Consider any $u \in \text{Reach}(x, S \cup \{y\})$. Clearly $u \neq y$. If u can be reached from x via S, it follows from lemma 4.4 that $u \in \text{Reach}(y,S)$. Otherwise, the path from x to u via $S \cup \{y\}$ has to go through the node y; again this implies $u \in \text{Reach}(y,S)$. □

Lemma 4.6 If $x \in \text{Reach}(y,S)$, then

$$\text{Reach}(x, S \cup \{y\}) \supset \text{Reach}(y,S) - \{x\}.$$

Proof Consider any $u \in \text{Reach}(y,S) - \{x\}$. There exists a path $u, s_1, \ldots, s_t, y$ where $\{s_1, \ldots, s_t\} \subset S$. However, $x \in \text{Reach}(y,S)$, which implies there exists one from y to x through S,

$$y, \bar{s}_1, \ldots, \bar{s}_r, x.$$

By joining the two paths, we see that u is reachable from x through $S \cup \{y\}$. □

Theorem 4.7 Let $x \in X\text{-}S$. If

$$x \in \text{Reach}(y,S)$$

and

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

then

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

Proof It is a direct consequence of corollary 4.5 and lemma 4.6. □

The result in theorem 4.7 can be used to speed up the symbolic factorization algorithm. After the reachable set

$$\text{Reach}_{G_{i-1}}(\{x_i\}, C(S_{i-1}))$$

has been determined at step 2, the two conditions in theorem 4.7 can be tested for the node $\{x_{i+1}\}$ in G_{i-1}. If they are satisfied, we have immediately the reach set for $\{x_{i+1}\}$. This can be applied repeatedly to

$$\{x_{i+1}\}, \{x_{i+2}\}, \ldots$$

until one that violates either of the two conditions is encountered. Only then, the quotient graph transformation step is performed.

In this way, we need only to examine the adjacent sets $Adj(x_{i+1})$, $Adj(x_{i+2})$, ... in order to find the reachable sets

$$Reach(x_{i+1}, S_i),\ Reach(x_{i+2}, S_{i+1}),\ \ldots$$

Moreover, these reachable sets can be represented implicitly by that of x_i. Thus, a set

$$M^* = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$$

of representatives is defined naturally by the algorithm, where each x_{i_k} represents the immediately succeeding nodes until the next representative $x_{i_{k+1}}$.

Let $|R_L| = \sum_{x_i \in M^*} |\ Reach(x_i, S_{i-1})\ |$. It follows then that the complexity of the improved algorithm is $O(|E^A| + |R_L|)$. In the next section, we consider a practical example where the improvement is significant.

§5 An Example - Nested Dissection on an n×n Grid

To demonstrate the effectiveness of the algorithm in section 4, we consider the nested dissection ordering [3, 6] of the n by n regular grid. Figure 5.1 shows such an ordering on the 10×10 grid problem.

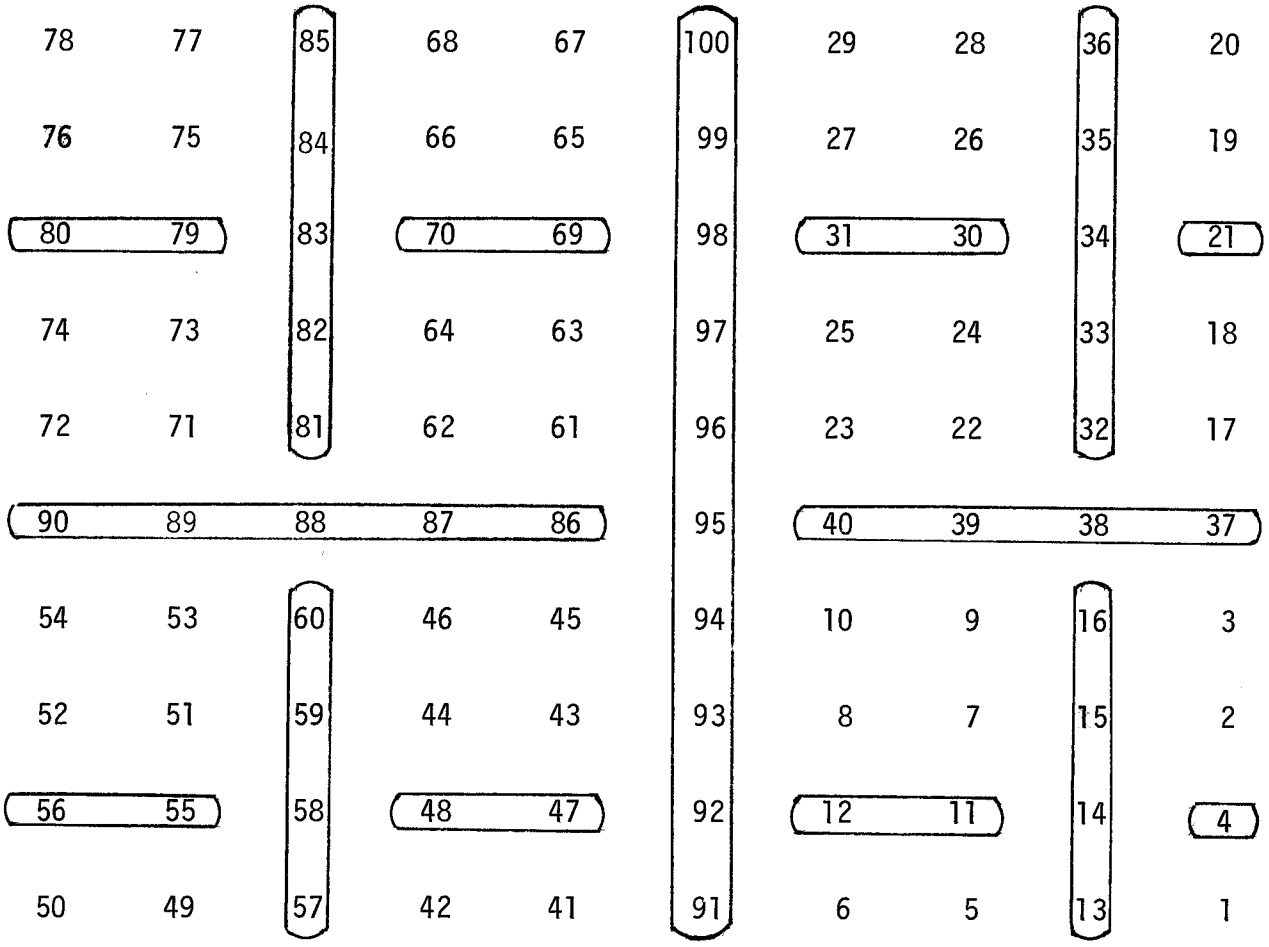

Figure 5.1 A nested dissection ordering of a 10×10 grid.

It has been established that the nested dissection strategy produces orderings that are optimal in the order of magnitude sense. With such orderings, the amount of arithmetic required to factor the matrix problem is $O(n^3)$ and the number of nonzeros in the factored matrix is $O(n^2 \log n)$.

We now consider the symbolic factorization of the grid problem with such orderings. On applying the improved algorithm in section 4, we note that the nodes in the "dissectors" (nodes grouped by incircling lines in figure 5.1) satisfy the conditions in theorem 4.7. Thus, as far as the symbolic factorization is concerned, a dissector can be represented by the lowest subscripted node in it, and such representation is in fact set up by the enhanced algorithm.

With the so-defined representative set, let s(n) be the number of nonzeros in the representative columns of the factored matrix for the n×n grid. That is s(n) is the corresponding size $|R_L|$. It is easy to establish the following recursive equations:

$$s(n) \le \bar{s}(n)$$

where $$\bar{s}(n) \le 12n + 4\bar{s}\left(\frac{n}{2}\right).$$

On solving the equations, we have

$$s(n) \le \bar{s}(n) \le 12n^2.$$

Together with the observation in section 4.3, we have proved that the symbolic factorization of an n by n grid problem ordered by nested dissection can be done in time proportional to n^2, even though the number of nonzeros in the triangular factor is $O(n^2 \log n)$.

§6. Some Numerical Experiments and Concluding Remarks

In this section we present numerical experiments demonstrating the performance of our algorithm. We also discuss its advantages and disadvantages compared to a very good "conventional" implementation supplied in the Yale Sparse Matrix Package [2], which was kindly provided to us by Professor Stanley Eisenstat. In what follows, we refer to the Yale routine as SSF, and to ours as SFQG.

In order to demonstrate experimentally the results of section 4, we applied SFQG to a sequence of "graded L" problems taken from [6]. This is a set of similar problems of increasing size, typical of those arising in finite element applications. The ordering used was produced by the implementation of the minimum degree algorithm described in [4]. Execution times reported are in seconds on an IBM 360/75 computer. The program was written in Fortran, and the optimizing version of the H-level compiler was used. The results of the experiment are summarized in Table 6.1. As the theory developed in section 4 predicts, the execution time for SFQG appears to be proportional to $|R_L|$.

How does SFQG compare with a good conventional implementation of symbolic factorization, such as the subroutine SSF from [2]? To a substantial degree, their relative merits depend upon the computing environment.

We have already observed that the major advantage of our subroutine is that it "fails gracefully". Since it does not use its output during execution, if insufficient storage is available, the output can be discarded butthe program can still execute, producing the number $|R_L|$. Alternatively, the output R_L could be printed on an auxiliary file as it is generated, and the subroutine then only needs storage for G^A.

N	Time	$\|R_L\|$	Time/$\|R_L\|$
265	.16	1353	1.19
406	.24	2252	1.08
577	.33	3293	1.05
778	.48	4604	1.04
1009	.58	5842	.99
1270	.80	7856	1.02
1560	.95	9424	1.00
1882	1.19	11707	1.01
			($x10^{-4}$)

Table 6.1 Execution time and $|R_L|$ for SFQG, along with their ratio, for the sequence of graded-L problems from [4].

To be fair, we must point out that in some contexts this storage argument is not relevant. Some ordering algorithms can be quite easily modified so as to provide either $|R_L|$ or a good upper bound, so that we can be assured when we use either SSF or SFQG with such orderings that they will not fail. However, it is not clear that all ordering algorithms can efficiently provide a good upper bound for $|R_L|$; for example, we do not know how to appropriately modify the automatic nested dissection algorithm in [6] so that an inexpensive estimate for $|R_L|$ is provided.

In terms of execution times our implementation is somewhat slower than SSF for small problems. The distributed version of SSF, applied to the graded-L problems of Table 6.1, produced the times shown in Table 5.2, along with which we have included the SFQG times from Table 5.1. After discussing our work with Professor Eisenstat, he showed us how to modify SSF so that it too employed our "mass elimination" technique, described in section 4.3. The column in Table 5.2 labelled SSF* contains the execution times of this modified subroutine.

The execution time of SSF is apparently growing faster than $|R_L|$, so for large enough problems, the SFQG subroutine would be faster than SSF. However, the improved version SSF* appears to execute in time proportional to $|R_L|$, and continues to enjoy a substantial execution time advantage over SFQG, even for the larger problems. We should note that neither SSF nor SSF*, as implemented, can be proved to run in $O(|R_L|)$ time. Modifications to these subroutines which allow the complexity bounds to be established considerably increase their execution times.

N	SFQG		SSF		SSF*	
	TIME	TIME / $\lvert R_L \rvert$	TIME	TIME / $\lvert R_L \rvert$	TIME	TIME / $\lvert R_L \rvert$
265	.16	1.19	.01	.07	.08	.06
406	.24	1.08	.17	.75	.14	.62
577	.33	1.05	.26	.78	.20	.60
778	.48	1.04	.37	.80	.27	.59
1009	.58	.99	.47	.80	.35	.60
1270	.80	1.02	.64	.81	.46	.59
1561	.95	1.00	.82	.87	.57	.60
1882	1.19	1.01	1.04	.88	.68	.58
		$(x10^{-4})$		$(x10^{-4})$		$(x10^{-4})$

Table 5.2 Comparison of execution times of SFQG, SSF and a modified version of SSF.

Thus, to summarize, our implementation of symbolic factorization, based on quotient graphs has some advantages in terms of storage and being able to "fail gracefully". In particular, since its execution is independent of its output, it is very attractive when storage is scarce, and **auxiliary storage devices must be used. In exchange, it appears to execute more slowly than the best conventional implementation of which** we are aware.

It is important for the reader to understand the sense in which our implementation is "optimal". We have shown only that it is optimal in the sense that it executes in time proportional to the size of its output, but we have not shown that its output is optimal. For purposes of data structure construction, it is desirable to produce an $\mathcal{R}_L$ having as few members as possible because this will tend to reduce the "overhead" storage requirements of the data structure for L. It is not difficult to construct examples where the $|\mathcal{R}_L|$ produced by our algorithm is larger than necessary. (In particular, note that SFQG only generates monotone representative maps M.) Investigation into the development of an algorithm which is optimal in both of the above senses is an interesting area of future research.

§7. References

[1] C. Berge, The Theory of Graphs and its Applications, John Wiley and Sons, New York, 1962.

[2] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman, "Yale Sparse Matrix Package I - The Symmetric Codes", Research Report #112, Dept. of Computer Science, Yale University, 1977.

[3] Alan George, "Numerical experiments using dissection methods to solve n by n grid problems", SIAM J. Numer. Anal., 14 (1977), pp. 161-180.

[4] Alan George and Joseph W.H. Liu, "A minimal storage implementation of the minimum degree algorithm", SIAM J. Numer. Anal., to appear.

[5] Alan George and Joseph W.H. Liu, "A quotient graph model for symmetric factorization", Research Report CS-78-04, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, February 1978.

[6] Alan George and Joseph W.H. Liu, "An automatic nested dissection algorithm for irregular finite element Problems", SIAM J. Numer. Anal., 15 (1978), pp. 1053-1069.

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

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

[9] A.H. Sherman, "On the efficient solution of sparse systems of linear and nonlinear equations", Research Rept. #46, Dept. of Computer Science, Yale University, 1975 (Doctoral Thesis).