FINDING DIAGONAL BLOCK ENVELOPES OF TRIANGULAR FACTORS OF PARTITIONED MATRICES+

by

Alan George*

and

Joseph W-H Liu**

Research Report CS-80-35

July 1980

* Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

** SBS Computer Shoppe
82 University Avenue
Toronto, Ontario, Canada M5J 1T5

+ Research supported in part by the Canadian Natural Sciences and Engineering Research Council under grant A8111.

ABSTRACT

Let A be a partitioned sparse symmetric positive definite matrix, and let L be its Cholesky factor, correspondingly partitioned. An important problem which arises in connection with allocating computer storage for L is to determine the envelope structure of the diagonal blocks of L, given the structure of A. The envelopes of A and L are known to be identical, because fill-in occurs only within the envelope, but the envelopes of their diagonal blocks in general differ. In this paper we provide an efficient algorithm for finding the envelopes of the diagonal blocks of L.

§1. Introduction

Let A be an N by N sparse symmetric positive definite matrix having a Cholesky factor L, where $A = LL^T$. For the i-th row of A, $i = 1,2,\ldots,N$, let

(1.1) $$f_i(A) = \min\{j \mid a_{ij} \neq 0\}.$$

The envelope of A, denoted by $Env(A)$, is then defined by

(1.2) $$Env(A) = \{\{i,j\} \mid f_i(A) \leq j < i\}.$$

This notion is important because in some contexts, computer storage methods involving the envelope of the matrix are very efficient [2,8]. Another attractive feature is that it can be shown [4] that $Env(A) = Env(L+L^T)$, so it is easy to determine the envelope of L from the structure of A. In what follows we denote the matrix sum $L + L^T$ by F, and refer to F as the filled matrix corresponding to A.

Suppose A is p by p symmetrically partitioned as shown in (1.3) and (1.4).

(1.3) $$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{12}^T & A_{22} & \cdots & A_{2p} \\ \vdots & & & \\ A_{1p}^T & A_{2p}^T & \cdots & A_{pp} \end{bmatrix}$$

The block diagonal matrix of A with respect to the given partitioning is defined to be

(1.4) $$Bdiag(A) = \begin{bmatrix} A_{11} & & & O \\ & A_{22} & & \\ & & \ddots & \\ O & & & A_{pp} \end{bmatrix}$$

Let the triangular factor L of A be correspondingly partitioned as

$$(1.5) \qquad L = \begin{bmatrix} L_{11} & & & 0 \\ L_{21} & L_{22} & & \\ \vdots & & \ddots & \\ L_{p1} & L_{p2} & \cdots & L_{pp} \end{bmatrix}$$

Then the associated block diagonal matrix of the filled matrix F will be

$$(1.6) \qquad \mathrm{Bdiag}(F) = \begin{bmatrix} F_{11} & & & 0 \\ & F_{22} & & \\ & & \ddots & \\ 0 & & & F_{pp} \end{bmatrix}$$

where $F_{kk} = L_{kk} + L_{kk}^T$, for $1 \le k \le p$.

In this paper, we consider the problem of determining the envelope structure of Bdiag(F), given the structure of A. We are motivated to consider this problem because some important storage schemes for the partitioned L require a knowledge of the envelopes of its diagonal submatrices [3,5].

§2. Graph Theory and Preliminary Results

The envelope structure of Bdiag(F) can be best studied using a graph theoretic approach. The reader is assumed to be familiar with the basic terminologies of graph theory, reference to which can be found in [1].

Let A be an N by N symmetric matrix. The labelled graph associated with A, denoted by $G^A = (X^A, E^A)$, is one for which the N nodes are labelled from 1 to N and $\{x_i, x_j\} \in E^A \iff a_{ij} = a_{ji} \neq 0$, $i \neq j$. Here x_i denotes the node of X^A with label i.

The function $f_i(A)$ in (1.2) can be expressed in terms of the graph G^A simply as

$$f_i(A) = \min\{j \mid x_j \in \mathrm{Adj}(x_i) \cup \{x_i\}\}.$$

To determine this value, we need only to inspect the adjacent set of the node x_i.

In order to study our problem posed in section 1, we first associate graphs with partitioned matrices. Let A be an p by p symmetrically partitioned matrix. Corresponding to the partitioning of the rows and columns of A, we associate a set partitioning of X^A.

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

Let F be the filled matrix of A. The envelope structure of Bdiag(F) is completely determined by the numbers $f_i(\mathrm{Bdiag}(F))$, that is, the column index of the first nonzero in each row of Bdiag(F). To determine these numbers, it is helpful to establish a relationship between the structures of G^A and G^F. The notion of reachable sets [6] serves this purpose.

Consider the graph $G = (X,E)$. Let S be a subset of the node set X with $x \notin S$. The node x is said to be reachable from a node y through S if there exists a path $(y,v_1,\ldots,v_k,x)$ from y to x such that $v_i \in S$ for $1 \le i \le k$. Note that k can be zero, so that any adjacent node of y not in S is reachable from y through S.

The reachable set of y through S, denoted by $Reach(y,S)$, is then defined to be

(2.1) $$Reach(y,S) = \{x \notin S \mid x \text{ is reachable from } y \text{ through } S\}.$$

The significance of the notion of reachable sets is embodied in the following theorem, which provides a relationship between the sets E^A and E^F in terms of reachable sets. The proof is given in [9] and is omitted.

Theorem 2.1

$$E^F = \{\{x_i,x_j\} \mid x_j \in Reach(x_i,\{x_1,x_2,\ldots,x_{i-1}\})\}.$$

□

In terms of the matrix, the set $Reach(x_i,\{x_1,\ldots,x_{i-1}\})$ is simply the set of row subscripts that correspond to nonzero entries in the column vector L_{*i}. (For more details, the reader is referred to [6].

§3. Main Results

In what follows, we shall use f_i to stand for $f_i(\text{Bdiag}(F))$. Let row i belong to the k-th block in the partitioning; in other words, we let $x_i \in Y_k$. In terms of the filled graph, the quantity f_i is given by

$$f_i = \min\{s \mid s = i \text{ or } \{x_s, x_i\} \in E^F(Y_k)\}.$$

We now relate it to the original graph G^A through the use of reachable sets introduced in Section 2. By Theorem 2.1 which characterizes the fill via reachable sets, we have

$$(3.1) \qquad f_i = \min\{s \mid x_s \in Y_k,\ x_i \in \text{Reach}(x_s,\ x_1, \ldots, x_{s-1}\}) \cup \{x_s\}\}.$$

In Theorem 3.2 below, we prove a somewhat stronger result. We begin with a lemma.

<u>Lemma 3.1</u> Let $x_i \in Y_k$, and let

$$S = Y_1 \cup \ldots \cup Y_{k-1}.$$

That is, S contains all the nodes in the first k-1 blocks. Then

$$x_i \in \text{Reach}(x_{f_i}, S) \cup \{x_{f_i}\}.$$

<u>Proof</u> By the definition of $f_i, \{x_i, x_{f_i}\} \in E^F$, so that by Theorem 5.1.2, $x_i \in \text{Reach}(x_{f_i}, \{x_1, \ldots, x_{f_i-1}\})$. We can then find a path $(x_i, x_{r_1}, \ldots, x_{r_t}, x_{f_i})$ where $\{x_{r_1}, \ldots, x_{r_t}\} \subset \{x_1, \ldots, x_{f_i-1}\}$.

We now prove that x_i can also be reached from x_{f_i} through S, which is a subset of $\{x_1, \ldots, x_{f_i-1}\}$. If $t = 0$, clearly $x_i \in \text{Reach}(x_{f_i}, S)$. On the other hand, if $t \neq 0$, let x_{r_s} be the node with the largest index number in $\{x_{r_1}, \ldots, x_{r_t}\}$. Then $(x_i, x_{r_1}, \ldots, x_{r_s-1}, x_{r_s})$ is a path from x_i to x_{r_s} through $\{x_1, x_2, \ldots, x_{r_s-1}\}$ so that

$$\{x_i, x_{r_s}\} \in E^F.$$

But $r_s < f_i$, so by the definition of f_i we have $x_{r_s} \notin Y_k$, or in other words $x_{r_s} \in S$. The choice of r_s implies

$$\{x_{r_1}, \ldots, x_{r_t}\} \subset S$$

and thus $x_i \in \text{Reach}(x_{f_i}, S)$.

□

<u>Theorem 3.2</u> Let $x_i \in Y_k$ and $S = Y_1 \cup \ldots \cup Y_{k-1}$. Then $f_i = \min\{s \mid x_s \in Y_k, x_i \in \text{Reach}(x_s, S) \cup \{x_s\}\}$.

<u>Proof</u> By Lemma 3.1, it remains to show that $x_i \notin \text{Reach}(x_r, S)$ for $x_r \in Y_k$ and $r < f_i$. Assume for contradiction that we can find $x_r \in Y_k$ with $r < f_i$ and $x_i \in \text{Reach}(x_r, S)$. Since

$$S \subset \{x_1, \ldots, x_{r-1}\},$$

we have $x_i \in \text{Reach}(x_r, \{x_1, \ldots, x_{r-1}\})$ so that $\{x_i, x_r\} \in E^F(Y_k)$. This contradicts the definition of f_i.

□

<u>Corollary 3.3</u> Let x_i and S be as in Theorem 3.2. Then

$$f_i = \min\{s \mid x_s \in \text{Reach}(x_i, S) \cup \{x_i\}\}.$$

□

The proof follows directly from Theorem 3.2 and the symmetry of the "Reach" operator. It is interesting to compare this result with that given by (3.1).

To illustrate the result, we consider the partitioned matrix example in Figure 3.1.

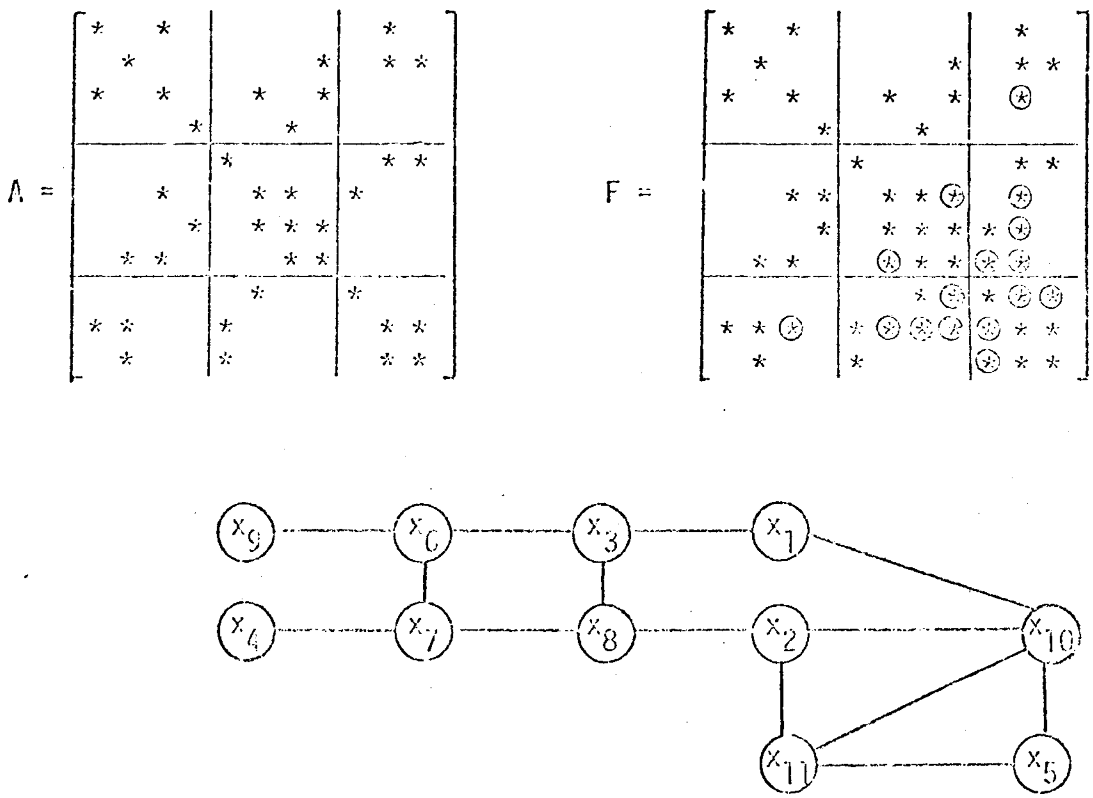

Figure 3.1 An 11×11 partitioned matrix A.

Consider $Y_2 = \{x_5, x_6, x_7, x_8\}$. Then the associated set $S = \{x_1, x_2, x_3, x_4\}$. We have

$$\text{Reach}(x_5, S) = \{x_{10}, x_{11}\}$$

$$\text{Reach}(x_6, S) = \{x_7, x_8, x_9, x_{10}\}$$

$$\text{Reach}(x_7, S) = \{x_6, x_8\}$$

$$\text{Reach}(x_8, S) = \{x_6, x_7, x_{10}, x_{11}\}.$$

By Corollary 3.3,

$$f_5(\text{Bdiag}(F)) = 5$$

$$f_6(\text{Bdiag}(F)) = f_7(\text{Bdiag}(F)) = f_8(\text{Bdiag}(F)) = 6.$$

§4. An Algorithm and Execution Time Analysis

Corollary 3.3 readily provides a method for finding $f_i(\mathrm{Bdiag}(F))$ and hence the envelope structure of Bdiag(F). However, in the actual implementation, Lemma 3.1 is more easily applied. Our algorithm can be described as follows.

Let $P = \{Y_1,\ldots,Y_p\}$ be the partitioning. For each block k in the partitioning, do the following:

Step 1 (Initialization) $S \leftarrow Y_1 \cup \ldots \cup Y_{k-1}$

$T \leftarrow S \cup Y_k$.

Step 2 (Main Loop) For each node x_r in Y_k do:

2.1) Determine $\mathrm{Reach}(x_r,S)$ in the subgraph $G(T)$.

2.2) For each $x_i \in \mathrm{Reach}(x_r,S)$, $f_i = r$.

2.3) Reset $T \leftarrow T - (\mathrm{Reach}(x_r,S) \cup \{x_r\})$.

Reset $S \leftarrow S - \{s \in S \mid s$ is reachable from x_r through a subset of $S\}$.

2.4) If $T = S$ then stop.

Step 2.3 needs some elaboration. For each node in $\mathrm{Reach}(x_r,S)$, its first envelope subscript is given by r. Once determined, we can remove these nodes from the set T in step 2.3. Furthermore, those nodes in S that have been traversed in finding $\mathrm{Reach}(x_r,S)$ can also be removed from S, since they cannot lead to new reachable nodes in T. These observations are important in obtaining the following execution time bound.

Theorem 4.1

Let $G = (X,E)$ and $P = \{Y_1,Y_2,\ldots,Y_p\}$ be a partitioning of X. Then the complexity of the algorithm described above is $O(p|E|)$.

Proof In performing the algorithm for the k-th block Y_k, the nodes and their neighbors in $Y_1 \cup Y_2 \cup \ldots \cup Y_k$ are inspected at most once. The upper bound $O(p|E|)$ then follows from summing over all p blocks.

□

Of course the actual running time of the algorithm depends on the way the blocks are connected, in addition to the quantities p and $|E|$ appearing in Theorem 4.1. In many realistic situations, the execution time is of lower order than that given in Theorem 4.1. For example, for so-called one-way dissection orderings [3], the bound is $O(|E|)$.

§5. An Algorithm for Finding an Approximate Diagonal Block Envelope

In most applications, it is not necessary to obtain the exact envelope structure of the diagonal blocks. An approximate structure is acceptable as long as it comes reasonably close to the exact envelope. In this section we provide an algorithm which is faster and simpler than the one described in Section 4, and for some applications usually provides the exact solution.

Let $P = \{Y_1, Y_2, \ldots, Y_p\}$ be the given partitioning. Let $x_i \in Y_k$ and $S = Y_1 \cup Y_2 \cup \ldots \cup Y_{k-1}$. As in previous sections, we denote

$$f_i = f_i(\mathrm{Bdiag}(F)).$$

Lemma 5.1 Either $x_{f_i} \in \mathrm{Adj}(x_i)$ or $\mathrm{Adj}(x_i) \cap S \neq \phi$ and $\mathrm{Adj}(x_{f_i}) \cap S \neq \phi$.

Proof By Corollary 3.3,

$$f_i = \min\{s \mid x_s \in \mathrm{Reach}(x_i, S) \cup \{x_i\}\}.$$

Assume $i \neq f_i$. Then $x_{f_i} \in \mathrm{Reach}(x_i, S)$; that is, there is a path

$$(x_i, s_1, \ldots, s_t, x_{f_i}).$$

If $t = 0$, then $x_{f_i} \in \mathrm{Adj}(x_i)$. Otherwise,

$$\mathrm{Adj}(x_i) \cap S = \phi$$

and

$$\mathrm{Adj}(x_{f_i}) \cap S = \phi.$$

□

Experience shows that in numerous situations, the converse of Lemma 5.1 provides a good approximate to the block envelope structure. For $x_i, x_j \in Y_k$, if

$$\mathrm{Adj}(x_i) \cap S \neq \phi$$

and

$$\mathrm{Adj}(x_j) \cap S \neq \phi,$$

then we assume $\{x_i, x_j\}$ is in the envelope structure of the diagonal blocks. The algorithm is as follows.

For each block k in the partitioning, do the following.

Step 1 (Initialization) $i_o \leftarrow N$, $S \leftarrow Y_1 \cup Y_2 \cup \ldots \cup Y_{k-1}$.

Step 2 (Main Loop) For each node $x_i \in Y_k$ do:

2.1) $f_i = \min\{s \mid x_s \in Y_k \cap (Adj(x_i) \cup \{x_i\})\}$.

2.2) $f_i \leftarrow \min\{f_i, i_o\}$.

2.3) If $Adj(x_i) \cap S \neq \phi$ then

$i_o \leftarrow \min\{i_o, i\}$.

It is obvious that this algorithm can be implemented to run in $O(|E|)$ time. In the next section, we shall consider some experimental results comparing the performance of this algorithm with that of Section 4.

§6. Numerical Experiments and Concluding Remarks

In this section we report on some numerical experiments designed to show the performance of the algorithm described in the previous two sections, and to support the claims made there. The Fortran implementations of the algorithms of Section 4 and 5 are referred to as FNBENV and FNTENV respectively in the tables which follow.

As a feasible alternative for solving this block envelope problem, we could simply apply a standard symbolic factorization procedure to the whole matrix, obtaining the entire structure of L; we could then easily obtain from this the block envelope structure. The state of the art for the general problem has reached a high level, so as a basis for timing comparison, we have included the times for a full symbolic factorization of the matrix problem in our tables. The name of the computer subroutine is SMBFCT; it is the fastest one we know about. Listings of this subroutine, as well as those for FNBENV and FNTENV, can be found in [7]. Of course we should point out that SMBFCT usually requires much more storage than either of the other two. Another point to remember is that the SMBFCT times reported should be regarded as lower bounds on the time required to solve the block envelope problem. We have not included time that would be needed to find the block envelope structure from the full structure of L provided by SMBFCT.

We used the set of "graded-L" mesh problems from [6, page 1062], which consists of a sequence of similar problems typical of those arising in finite element applications. We used two different ordering algorithms, both of which produce partitionings of the correspondingly ordered matrices. These are the one-way dissection ordering algorithm (1WD) [3], and the refined quotient tree ordering (RQT) [5]. In both cases the diagonal blocks of the partitioned matrices tend to be sparse, but also tend to have full envelopes. Thus, it makes sense to use a storage scheme which exploits this fact, and in order to set up such a data structure we must solve the problem addressed in this paper. The results of the experiments are summarized in Tables 6.1 and 6.2. Execution times are in seconds on an IBM 3031.

Figure 6.1 shows that the approximate algorithm of Section 5 can pay handsomely. Note that its execution time appears to be $O(|E|)$ as expected, while both FNBENV and SMBFCT appear to be $O(p|E|)$. The reason that SMBFCT appears to be $O(p|E|)$ has nothing to do with the partitioning per se, since the algorithm does not use it in any way. The apparent connection is due to the fact that for these problems, we expect the execution time of SMBFCT to be $O(|E|^{3/2})$, and we also expect the RQT algorithm to yield $p \approx \frac{1}{2}|E|^{\frac{1}{2}}$.

Table 6.2 shows that FNTENV is no panacea; for some problems it fails to find an envelope that is acceptably close to the correct one. The data in the table also shows that FNBENV may execute much faster than the bound provided by Theorem 4.1. It can be shown that for one-way dissection orderings, FNBENV executes in $O(|E|)$ time [7], and the numbers in the fourth column of Table 6.1 certainly support this result. Note that FNBENV executes substantially faster than SMBFCT for these problems.

Ideally, we would like an algorithm which combines those of Section 4 and 5, so that the "short cut" scheme of Section 5 is used only when it is applicable, and the algorithm of Section 4 is used otherwise. So far we have not discovered a cheap test to indicate that FNTENV is inadequate for a problem, although one may exist. In any case, we feel that FNBENV provides an acceptable solution to our problem, even when it may be somewhat slower than the full symbolic factorization SMBFCT approach. One important practical disadvantage of the latter approach is that storage requirements are unpredictable, while FNBENV uses a fixed predictable amount of storage consisting of only a few arrays of length N, in addition to that required for the graph of A.

			FNBENV			FNTENV		SMBFCT		
N	$\|E\|$	P	TIME	TIME/$\|E\|$	TIME/$(p\|E\|)$	TIME	TIME/$\|E\|$	TIME	TIME/$\|E\|$	TIME/$(p\|E\|)$
265	774	25	.22	3.00	1.20	.04	5.37	.12	1.57	.62
406	1155	31	.41	3.58	1.15	.06	5.48	.21	1.79	.58
577	1656	37	.68	4.13	1.12	.10	5.83	.32	1.93	.52
778	2247	43	1.04	4.64	1.08	.13	5.63	.49	2.16	.50
1009	2928	49	1.52	5.20	1.06	.16	5.35	.69	2.37	.48
1270	3699	55	2.12	5.72	1.04	.20	5.31	.97	2.62	.48
1561	4560	61	2.89	6.34	1.04	.25	5.41	1.26	2.77	.45
1882	5511	67	3.75	6.81	1.02	.29	5.32	1.66	3.01	.45
2233	6552	73	4.82	7.36	1.01	.36	5.44	2.11	3.22	.44
				$(\times 10^{-4})$	$(\times 10^{-5})$		$(\times 10^{-5})$		$(\times 10^{-4})$	$(\times 10^{-5})$

Table 6.1 Execution times for FNBENV, FNTENV, and SMBFCT, applied to a sequence of problems ordered and partitioned by the RQT algorithm from [5]. In all cases, FNTENV found the exact envelope.

N	|E|	P	FNBENV TIME	FNBENV TIME/|E|	FNBENV ENVELOPE SIZE	FNTENV TIME	FNTENV ENVELOPE SIZE	SMBFCT TIME	SMBFCT TIME/|E|
265	744	6	.07	9.41	1731	.04	2011	.12	1.65
406	1155	6	.11	9.52	3103	.05	3760	.20	1.76
577	1656	7	.16	9.46	5091	.08	6761	.33	2.00
778	2247	8	.21	9.34	7401	.11	10420	.49	2.20
1009	2928	8	.28	9.45	10239	.14	14958	.70	2.38
1270	3699	9	.35	9.46	13832	.18	21497	.96	2.60
1561	4560	9	.43	9.50	18374	.25	29633	1.27	2.78
1882	5511	10	.51	9.31	23081	.27	39478	1.65	2.98
2233	6552	10	.61	9.26	28966	.31	50318	2.07	3.16
				$(\times 10^{-5})$					$(\times 10^{-4})$

Table 6.2 Execution times for FNBENV, FNTENV, and SMBFCT, applied to a sequence of problems from [6], ordered and partitioned by the one-way dissection algorithm from [3]. The envelope sizes found by FNBENV and FNTENV show that the approximate method used by FNTENV is not adequate for these problems.

§7. References

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

[2] C. A. Felippa, "Solution of linear equations with skyline-stored matrix", Computers and Structures 5 (1975), pp. 12-39.

[3] Alan George, "An automatic one-way dissection algorithm for irregular finite element problems", Proc. 1977 Dundee Conf. on Numerical Analysis, Lecture Notes No. 630, Springer Verlag, 1978, pp. 76-89.

[4] Alan George and Joseph W-H Liu, "A note on fill for sparse matrices", SIAM J. Numer. Anal. 12 (1975), pp. 452-455.

[5] Alan George and Joseph W-H Liu, "Algorithms for matrix partitioning and the numerical solution of finite element systems", SIAM J. Numer. Anal. 15 (1978), pp. 297-327.

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

[7] Alan George and Joseph W-H Liu, Computer Solution of Large Positive Definite Systems, to be published by Prentice-Hall Inc.

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

[9] D. J. Rose, R. E. Tarjan, and G. S. Lueker, "Algorithmic aspects of vertex elimination on graphs", SIAM J. on Computing 5 (1976), pp. 266-283.

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WATERLOO
TECHNICAL REPORTS 1979

Report No.	Author	Title
CS-79-01*	E.A. Ashcroft W.W. Wadge	Generality Considered Harmful - A Critique of Descriptive Semantics
CS-79-02*	T.S.E. Maibaum	Abstract Data Types and a Semantics for the ANSI/SPARC Architecture
CS-79-03*	D.R. McIntyre	A Maximum Column Partition for Sparse Positive Definite Linear Systems Ordered by the Minimum Degree Ordering Algorithm
CS-79-04*	K. Culik II A. Salomaa	Test Sets and Checking Words for Homomorphism Equivalence
CS-79-05*	T.S.E. Maibaum	The Semantics of Sharing in Parallel Processing
CS-79-06*	C.J. Colbourn K.S. Booth	Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs
CS-79-07*	K. Culik, II N.D. Diamond	A Homomorphic Characterization of Time and Space Complexity Classes of Languages
CS-79-08*	M.R. Levy T.S.E. Maibaum	Continuous Data Types
CS-79-09	K.O. Geddes	Non-Truncated Power Series Solution of Linear ODE's in ALTRAN
CS-79-10*	D.J. Taylor J.P. Black D.E. Morgan	Robust Implementations of Compound Data Structures
CS-79-11*	G.H. Gonnet	Open Addressing Hashing with Unequal-Probability Keys
CS-79-12	M.O. Afolabi	The Design and Implementation of a Package for Symbolic Series Solution of Ordinary Differential Equations
CS-79-13*	W.M. Chan J.A. George	A Linear Time Implementation of the Reverse Cuthill-McKee Algorithm
CS-79-14	D.E. Morgan	Analysis of Closed Queueing Networks with Periodic Servers
CS-79-15*	M.H. van Emden G.J. de Lucena	Predicate Logic as a Language for Parallel Programming
CS-79-16*	J. Karhumäki I. Simon	A Note on Elementary Homorphisms and the Regularity of Equality Sets
CS-79-17*	K. Culik II J. Karhumäki	On the Equality Sets for Homomorphisms on Free Monoids with two Generators
CS-79-18*	F.E. Fich	Languages of R-Trivial and Related Monoids

* Out of print - contact author

Technical Reports 1979

CS-79-19*	D.R. Cheriton	Multi-Process Structuring and the Thoth Operating System
CS-79-20*	E.A. Ashcroft W.W. Wadge	A Logical Programming Language
CS-79-21*	E.A. Ashcroft W.W. Wadge	Structured LUCID
CS-79-22	G.B. Bonkowski W.M. Gentleman M.A. Malcolm	Porting the Zed Compiler
CS-79-23*	K.L. Clark M.H. van Emden	Consequence Verification of Flow-charts
CS-79-24*	D. Dobkin J.I. Munro	Optimal Time Minimal Space Selection Algorithms
CS-79-25*	P.R.F. Cunha C.J. Lucena T.S.E. Maibaum	On the Design and Specification of Message Oriented Programs
CS-79-26*	T.S.E. Maibaum	Non-Termination, Implicit Definitions and Abstract Data Types
CS-79-27*	D. Dobkin J.I. Munro	Determining the Mode
CS-79-28	T.A. Cargill	A View of Source Text for Diversely Configurable Software
CS-79-29*	R.J. Ramirez F.W. Tompa J.I. Munro	Optimum Reorganization Points for Arbitrary Database Costs
CS-79-30	A. Pereda R.L. Carvalho C.J. Lucena T.S.E. Maibaum	Data Specification Methods
CS-79-31*	J.I. Munro H. Suwanda	Implicit Data Structures for Fast Search and Update
CS-79-32*	D. Rotem J. Urrutia	Circular Permutation Graphs
CS-79-33*	M.S. Brader	PHOTON/532/Set - A Text Formatter
CS-79-34*	D.J. Taylor D.E. Morgan J.P. Black	Redundancy in Data Structures: Improving Software Fault Tolerance
CS-79-35	D.J. Taylor D.E. Morgan J.P. Black	Redundancy in Data Structures: Some Theoretical Results
CS-79-36	J.C. Beatty	On the Relationship between the LL(1) and LR(1) Grammars
CS-79-37	E.A. Ashcroft W.W. Wadge	R_x for Semantics

* Out of print - contact author

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WATERLOO

RESEARCH REPORTS 1980

Report No.	Author	Title
CS-80-01	P.T. Cox T. Pietrzykowski	On Reverse Skolemization
CS-80-02	K. Culik II	Homomorphisms: Decidability, Equality and Test Sets
CS-80-03	J. Brzozowski	Open Problems About Regular Languages
CS-80-04	H. Suwanda	Implicit Data Structures for the Dictionary Problem
CS-80-05	M.H. van Emden	Chess-Endgame Advice: A Case Study in Computer Utilization of Knowledge
CS-80-06	Y. Kobuchi K. Culik II	Simulation Relation of Dynamical Systems
CS-80-07	G.H. Gonnet J.I. Munro H. Suwanda	Exegesis of Self-Organizing Linear Search
CS-80-08	J.P. Black D.J. Taylor D.E. Morgan	An Introduction to Robust Data Structures
CS-80-09	J.Ll. Morris	The Extrapolation of First Order Methods for Parabolic Partial Differential Equations II
CS-80-10*	N. Santoro H. Suwanda	Entropy of the Self-Organizing Linear Lists
CS-80-11	T.S.E. Maibaum C.S. dos Santos A.L. Furtado	A Uniform Logical Treatment of Queries and Updates
CS-80-12	K.R. Apt M.H. van Emden	Contributions to the Theory of Logic Programming
CS-80-13	J.A. George M.T. Heath	Solution of Sparse Linear Least Squares Problems Using Givens Rotations
CS-80-14	T.S.E. Maibaum	Data Base Instances, Abstract Data Types and Data Base Specification
CS-80-15	J.P. Black D.J. Taylor D.E. Morgan	A Robust B-Tree Implementation
CS-80-16	K.O. Geddes	Block Structure in the Chebyshev-Padé Table
CS-80-17	P. Calamai A.R. Conn	A Stable Algorithm for Solving the Multi-facility Location Problem Involving Euclidean Distances

CS-79-38	E.A. Ashcroft W.W. Wadge	Some Common Misconceptions about LUCID
CS-79-39	J. Albert K. Culik II	Test Sets for Homomorphism Equivalence on Context Free Languages
CS-79-40	F.W. Tompa R.J. Ramirez	Selection of Efficient Storage Structures
CS-79-41*	P.T. Cox T. Pietrzykowski	Deduction Plans: A Basis for Intelligent Backtracking
CS-79-42	R.C. Read D. Rotem J. Urrutia	Orientations of Circle Graphs

* Out of print - contact author

CS-80-18	R.J. Ramirez	Efficient Algorithms for Selecting Efficient Data Storage Structures
CS-80-19	D. Therien	Classification of Regular Languages by Congruences
CS-80-20	J. Buccino	A Reliable Typesetting System for Waterloo
CS-80-21	N. Santoro	Efficient Abstract Implementations for Relational Data Structures
CS-80-22	R.L. de Carvalho T.S.E. Maibaum T.H.C. Pequeno A.A. Pereda P.A.S. Veloso	A Model Theoretic Approach to the Theory of Abstract Data Types and Data Structures
CS-80-23	G.H. Gonnet	A Handbook on Algorithms and Data Structures
CS-80-24	J.P. Black D.J. Taylor D.E. Morgan	A Case Study in Fault Tolerant Software
CS-80-25	N. Santoro	Four 0(n**2) Multiplication Methods for Sparse and Dense Boolean Matrices
CS-80-26	J.A. Brzozowski	Development in the Theory of Regular Languages
CS-80-27	J. Bradford T. Pietrzykowski	The Eta Interface
CS-80-28	P. Cunha T.S.E. Maibaum	Resource = Abstract Data Type Data + Synchronization ...
CS-80-29	K. Culik II Arto Salomaa	On Infinite Words Obtained by Interating Morphisms
CS-80-30	T.F. Coleman A.R. Conn	Nonlinear Programming via an Exact Penalty Function: Asymptotic Analysis
CS-80-31	T.F. Coleman A.R. Conn	Nonlinear Programming via an Exact Penalty Function: Global Analysis
CS-80-32	P.R.F. Cunha C.J. Lucena T.S.E. Maibaum	Message Oriented Programming - A Resource Based Methodology
CS-80-33	Karel Culik II Tero Harju	Dominoes Over A Free Monoid
CS-80-34	K.S. Booth	Dominating Sets in Chordal Graphs
CS-80-35	A. George J. W-H Liu	Finding Diagonal Block Envelopes of Triangular Factors of Partitioned Matrices