An Automatic One-Way Dissection Algorithm for Irregular Finite Element Problems*

by

Alan George†

Research Report CS-77-06

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

April 1977

*To appear in the Proceedings of the 1977 Dundee Biennial Conference on Numerical Analysis. Research supported in part by Canadian National Research Council grant A8111.

Abstract

An algorithm for automatically finding a one-way dissection ordering for irregular finite element problems is described. Numerical experiments suggest that the amount of fill suffered by correspondingly ordered matrices, when factored, is $0(N^{5/4})$, and that the amount of arithmetic required to perform the factorization is $0(N^{7/4})$. The corresponding estimates for nested dissection orderings are $0(N \log N)$ and $0(N^{3/2})$ respectively, so the one-way scheme is asymptotically inferior. However, experiments suggest that unless N is very large indeed, the one-way dissection orderings require considerably less storage than the nested dissection orderings, although the arithmetic requirements are larger.

§1 Introduction

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

(1.1) $Ax = b,$

where A is a sparse N by N positive definite matrix arising in the application of finite element methods [15, 17]. The method we use is standard; the matrix A is factored in the product LL^T, where L is lower triangular, and then the triangular systems $Ly=b$ and $L^Tx=y$ are solved to obtain x.

When the matrix A is factored it usually suffers fill; that is, making the usual assumption that exact cancellation does not occur, $L+L^T$ is usually fuller than A. Since PAP^T is positive definite for any N by N permutation matrix P, Cholesky's method can be used to solve the equivalent system

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

It is well known that a judicious choice of P can often drastically reduce fill and/or arithmetic requirements.

Recently, the author has described two efficient orderings for the system of $N=n^2$ equations arising in connection with the use of finite difference or finite element methods on an n by n grid [6]. These schemes, called nested dissection and one-way dissection, are efficient in the sense that they reduce the arithmetic required to factor A to $0(N^{3/2})$ and $0(N^{7/4})$ respectively, compared to $0(N^2)$ if the usual row by row numbering of the grid is used. In addition, the amount of storage required if one uses these dissection strategies is $0(N \log N)$ and $0(N^{5/4})$, compared to $0(N^{3/2})$. In [6], it was demonstrated that by careful selection of data structures, these orderings could be utilized in linear equation solvers so that their

execution times and storage requirements were as the operation and fill counts suggest.

However, although simple model n by n grid problems are easy to analyze and interesting from a theoretical point of view, a much more desirable practical requirement is to find similarly efficient orderings for less regular problems. In [8], George and Liu have provided an automatic algorithm for producing nested dissection orderings for irregular finite element problems. The objective of this paper is to provide an automatic scheme for finding orderings analogous to the one-way dissection orderings. As we shall see later, the storage requirements for these orderings appear to grow as $N^{5/4}$, as we expect in view of the results for the n by n grid problem. On the surface, it appears that such orderings are inferior to nested dissection orderings, whose storage requirements only grow as $0(N \log N)$. However, the estimates are asymptotic, and unless N is very large indeed, the one-way dissection orderings appear to require considerably less storage than the nested dissection orderings. (This is also the situation for the n by n grid problem.) In exchange, for lower storage requirements, we normally perform more arithmatic than for the nested dissection orderings. Thus, the automatic determination of such one-way orderings for irregular problems is important when storage is limited. In this paper a heuristic algorithm is described for finding one-way dissection orderings for sparse matrix problems, and some numerical experiments describing its application to some finite element problems are provided.

The class of problems for which this one-way dissection algorithm is primarily intended are those arising in connection with the application of finite difference and finite element methods. For two dimensional problems, they can be characterized as follows. Let M be a planar mesh

consisting of triangles and/or quadrilaterals called elements, as shown in Figure 1.1, where adjacent elements have a common side or vertex. The mesh has nodes at each vertex, and there may also be nodes lying on element edges and faces; associated with each node is one or more variables x_i, and for some labelling of these N variables we define a finite element system $Ax = b$ associated with M as one for which A is symmetric and positive definite, and for which $A_{ij} \neq 0 \Rightarrow x_i$ and x_j are associated with the same node, or nodes belonging to the same element.

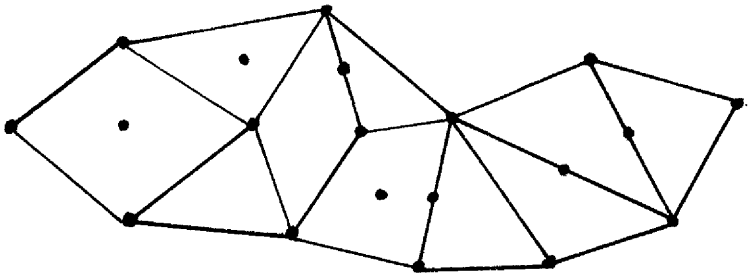

Figure 1.1 A 20 node finite element mesh with 10 elements.

An outline of the paper is as follows. In section 2 a simple model grid problem is analyzed to provide the motivation for the ordering algorithm. Section 3 contains a description of the actual ordering algorithm, and section 4 contains some numerical experiments with the algorithm, applied to problems typical of those arising in finite element applications. Finally, section 5 contains some concluding remarks.

§2 Motivation†

Consider an m by ℓ grid or mesh as shown in Figure 2.1, having $N=m\ell$ nodes. The corresponding matrix problem we consider has the property that for any numbering of the nodes from 1 to N, the coefficient matrix A satisfies $A_{ij} \neq 0$ => node i and node j belong to the same small square. This corresponds to the familiar 9 point difference applied to a regular discretization of a rectangular domain.

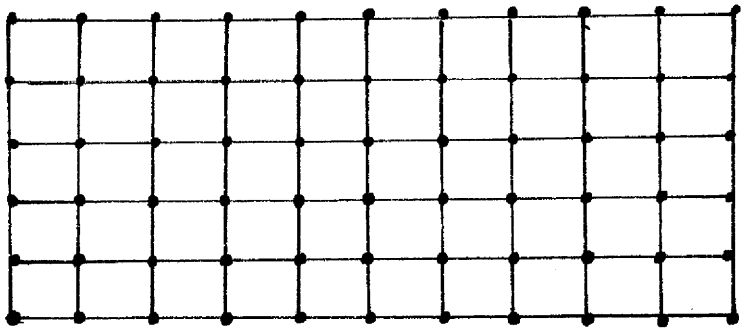

Figure 2.1 An m by ℓ grid with m=6 and ℓ=11.

Let α be an integer satisfying $1 \leq \alpha \leq \ell$, and choose α vertical grid lines (separators) which dissect the grid into α+1 independent blocks, as depicted in Figure 2.2, where α=3. The α+1 independent blocks are numbered row by row, followed by the α separators, as indicated in Figure 2.2. The matrix structure that this ordering induces is shown in Figure 2.3.

---

†This development closely parallels that in George [6,§4] except that here we treat the case for m≠ℓ. This is important as it provides a mechanism for choosing α when the mesh is irregular.

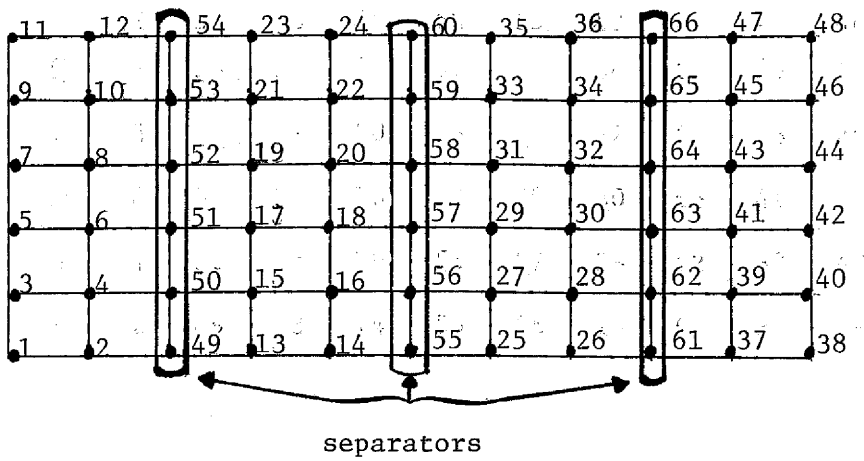

Figure 2.2 One-way dissection ordering of the grid of Figure 2.1, with the number of separators α equal to 3.

The keys to the efficient use of this ordering can be illustrated by considering the block 2 by 2 symmetric positive definite matrix

(2.1)
$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix},$$

whose (block) Cholesky factorization is

(2.2)
$$L = \begin{pmatrix} L_{11} & 0 \\ W_{12}^T & L_{22} \end{pmatrix},$$

where

(2.3)
$$A_{11} = L_{11}L_{11}^T ,$$

(2.4)
$$W_{12} = L_{11}^{-1} A_{12} ,$$

and

(2.5)
$$L_{22}L_{22}^T = \tilde{A}_{22} = A_{22} - A_{12}^T A_{11}^{-1} A_{12}.$$

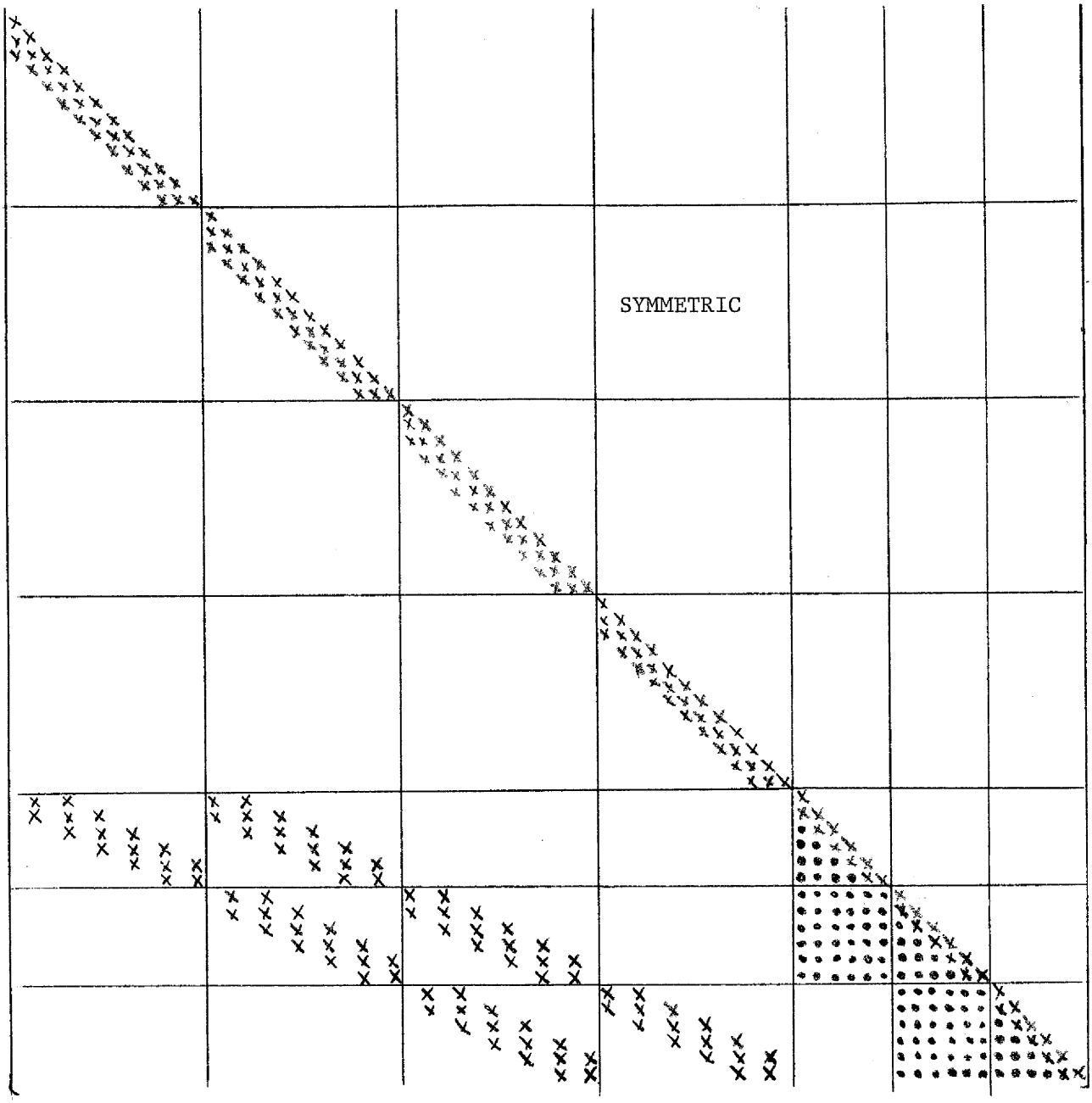

Figure 2.3 Matrix structure induced by the ordering of Figure 2.2. The dots indicate the fill suffered by the diagonal block corresponding to the separators.

As usual, it should be understood that inverses are not actually computed; instead, the appropriate triangular systems are solved.

The first observation is that $\tilde{A}_{22}$ can be computed either as $A_{22} - (A_{12}^T L_{11}^{-T})(L_{11}^{-1} A_{12}) = A_{22} - W_{12}^T W_{12}$, or as $A_{22} - A_{12}^T (L_{11}^{-T} (L_{11}^{-1} A_{12}))$. In general, for sparse matrices these factorizations require different amounts of computation. A particularly important point is that the second form of the computation does not require storage for W_{12}, since the computation can be carried out one column at a time. Hence only one auxiliary vector is required. On the other hand, the first computation appears to require the storage of the entire matrix W_{12}. See [5] for details.

The second important observation is that we may not wish to retain W_{12}. As observed by Bunch and Rose [3], it may require fewer arithmetic operations to operate only implicitly with W_{12}, by using (2.4). For example, to compute $\tilde{x}_2 = W_{12}x_2$ we can calculate $z = A_{12}x_2$, and then solve the triangular system $L_{11}\tilde{x}_2 = z$. If there are fewer nonzeros in total in L_{11} and A_{12} than in W_{12}, this mode of calculating will save arithmetic operations, and will also save primary storage, since A_{12} rather than W_{12} is stored. Here primary storage is that actually used to store numerical values, rather than storage for pointers etc, which we will call overhead storage.

Returning now to our one-way dissection ordering, it turns out to be crucial to use both observations, where the leading $\alpha+1$ blocks corresponding to the "independent blocks" are viewed as A_{11}, and the diagonal block corresponding to the α separators is A_{22}. Thus, the only part of L that we store will be the lower triangles of the $2\alpha+1$ diagonal blocks, along with the $\alpha-1$ $\hat{m}$ by m "fill" blocks directly below the last α diagonal blocks.

The computation of $\tilde{A}_{22}$ is performed in the asymmetric way, thus avoiding storing W_{12} at any time.

We now do a rough analysis of storage requirements in order to determine what value α should have for general m and ℓ. In the actual implementation the diagonal blocks A_{11} and A_{22} are treated as one matrix whose variation in bandwidth is exploited by a storage scheme similar to that proposed by Jennings [11]. The matrix A_{12} is very sparse, so only its nonzeros are stored. The important point here is that the total number of nonzeros in L_{11}, L_{22} and A_{12} is given approximately by

$$p(\alpha) = \frac{m(\ell-\alpha)^2}{(\alpha+1)} + \alpha\frac{m(m+1)}{2} + (\alpha-1)m^2 + 0(\alpha m,\ell). \tag{2.6}$$

Differentiating with respect to α, we find p is approximately minimized for

$$\tilde{\alpha} = \ell\{2/3(m+1)\}^{\frac{1}{2}} - 1, \tag{2.7}$$

yielding

$$p(\tilde{\alpha}) = \sqrt{3/2}\; m\ell\,\{\ell^{\frac{1}{2}}+m^{\frac{1}{2}}\} + 0(m\ell). \tag{2.8}$$

Of course in practice $\tilde{\alpha}$ must be chosen to be an integer, and the dissection in general will not be exactly uniform unless $(\ell-\alpha)\ /\ (\alpha+1)$ is also an integer. However, these minor observations turn out to be irrelevant from a practical viewpoint since $|p'(\alpha)|$ is small near $\tilde{\alpha}$.

Note that if we used an ordinary column-by-column numbering scheme we would obtain a band matrix having bandwidth $\beta = m+2$. (We define the bandwidth β of a matrix M to be $\max\{|i-j| \mid M_{ij} \neq 0\}$.) Using a standard band storage scheme [16], our storage requirements would be $\frac{1}{2}m^2\ell + 0(m\ell)$. Comparing with (2.8), we can conclude that our one-way dissection scheme will be preferable to the standard band scheme provided that $m \gtrsim \sqrt{6\ell}$. When m is proportional to ℓ, (2.8) implies that the one-way dissection scheme

requires $0(N^{5/4})$ storage, compared to $0(N^{3/2})$ for the standard band scheme. It is a straightforward exercise to show that the operation count for the factorization is $0(N^{7/4})$ compared to $0(N^2)$ for the standard band scheme.

This simple analysis provides us with some insight into the general nature of a one-way dissection ordering for irregular mesh problems such as the one depicted in Figure 1.1. We would like to find a set of separators, hopefully having relatively few nodes, which disconnect the mesh into pieces which can be numbered so that the corresponding matrices have a small bandwidth or envelope. In the next section we rephrase the ordering problem in graph theory terms and provide a heuristic algorithm for finding such separators and orderings.

§3 The Ordering Algorithm

3.1 Graph Theory Terminology

It is convenient to describe the ordering algorithm in terms of labelling an undirected graph, so we begin by introducing some standard graph theory notions. An undirected graph $G = (X,E)$ is a finite non-empty set X of nodes or vertices together with a set E of edges, which are ordered pairs of distinct nodes of X. A graph $G' = (X',E')$ is a subgraph of G if $X' \subseteq X$ and $E' \subseteq E$. For $Y \subset X$, the section graph $G(Y)$ is the subgraph $(Y,E(Y))$, where

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

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

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

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

The set $Y \subset X$ is a separator of the connected graph G if the section graph $G(X\backslash Y)$ is disconnected; Y is a minimal separator if no proper subset of Y is a separator of G.

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

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

where $\bigcup_{i=1}^{p} Y_i = X$ and $Y_i \cap Y_j = \emptyset$ for $i \neq j$. An ordering (labelling, numbering) of G is a bijective mapping $\alpha: \{1,2,3,\ldots,N\} \to X$, where $N=|X|$.

An important type of graph partitioning, which plays an integral role in our one-way dissection algorithm, is the class of level structures [1]. A level structure of a connected graph $G = (X,E)$ is a partitioning

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

of the node set X such that $Adj(L_0) \subset L_1$, $Adj(L_\ell) \subset L_{\ell-1}$, and $Adj(L_i) \subset L_{i-1} \cup L_{i+1}$ for $0 < i < \ell$. A rooted level structure, rooted at $x \in X$, is the level structure

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

where $L_0(x) = \{x\}$ and $L_i(x) = Adj\left(\bigcup_{j=0}^{i-1} L_j\right)$. The number $\ell(x)$ is sometimes called the eccentricity of x. The diameter of G is then defined by

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

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

The effectiveness of the ordering algorithm described later in this section hinges on finding a relatively "long, narrow" level structure. That is, one having relatively many levels, and relatively few nodes in each level. It is intuitively clear that a level structure rooted at a peripheral node would be a good candidate.

Unfortunately, no efficient algorithm is known to determine such nodes for a general graph. Recently Gibbs et. al. [10] have devised an algorithm for finding nodes of high eccentricity. A modification of this algorithm, described in [7], is used in the algorithm described in this paper. The root used for the level structure will be called a pseudo-peripheral node.

We now establish a connection between graphs and matrices. Let A be an N by N symmetric matrix. The labelled undirected graph of A, denoted by $G^A = (X^A, E^A)$, is one for which X^A is labelled from 1 to N and $\{x_i, x_j\} \in E^A$ if and only if $A_{ij} \neq 0$, $i > j$. The unlabelled graph of A is

simply G^A with its labels removed. For any N by N permutation matrix P, the unlabelled graphs of A and PAP^T are the same, but the associated labellings differ. Finding a good ordering for A can thus be viewed as finding a good labelling for its graph.

3.2 Description of the Algorithm

We begin with a step-by-step description of the algorithm, followed by some explanatory remarks for the more important steps.

1. Find a pseudo-peripheral node x, using the algorithm described in [7], and then generate the level structure $\mathcal{L}(x) = \{L_1, L_2, \ldots, L_\ell\}$ rooted at x.
2. (Estimate $\tilde{\alpha}$) Calculate $m = N/(\ell+1)$, and if $m \leq (6\ell)^{\frac{1}{2}}$, go to step 5. Otherwise set $\tilde{\alpha} = \ell(2/3(m+1))^{\frac{1}{2}} - 1$ and go to step 3.
3. Set $\delta = \ell/(\tilde{\alpha}+1)$. If $\delta \geq 2$, go to step 6. Otherwise go to step 4.
4. (Correct estimate for $\tilde{\alpha}$) Decrement $\tilde{\alpha}$ by 1, and if $\tilde{\alpha} > 1$, go to step 3. Otherwise go to step 5.
5. Set k=1, $Y_1 = X$, α=0, and then go to step 8.
6. (Find separators) Set $\alpha = \lfloor \tilde{\alpha} \rfloor$.

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

   6.1 Set $j = \lfloor i\,\delta - 1 \rfloor$.

   6.2 Choose $T_i \subseteq L_j$ to be a minimal separator of G.
7. Let Y_i, i=1,2,...,k be the connected components of the section graph $G(X \setminus \bigcup_{j=1}^{\alpha} T_j)$, and $Y_{k+j} = T_j$, j=1,2,...,α.
8. Number each Y_i, i=1,2,...,k consecutively using the reverse Cuthill-McKee (RCM) algorithm [12], and if $\alpha > 0$, number the Y_{k+j}, j=1,2,...,α consecutively in any order.

Step 1 of the algorithm produces the (hopefully) long, narrow level structure referred to in section 2. This is desirable because the separators

are selected as subsets of some of the levels L_i.

The calculation of the numbers m and $\tilde{\alpha}$ computed in step 2 is motivated directly by the crude analysis of the m by ℓ grid in section 2. Since m is the average number of nodes per level, it serves as a measure of the width of the level structure.

Steps 4 and 5 are normally not executed; they are designed to handle anomalous situations where $m << \ell$, or when N is simply too small to make the application of the dissection algorithm sensible. In these cases, the entire graph is processed as one block (α=0). That is, an ordinary band type ordering is produced for the graph.

Step 6 performs the actual selection of the separators, and is done essentially as though the graph corresponded to an m by ℓ grid as studied in section 2. As noted earlier, each L_i of $\mathcal{L}$ is a separator of G, although not necessarily minimal. In step 6, α approximately equally spaced levels are chosen from $\mathcal{L}$, and subsets of these levels (the T_i) which are minimal separators are then found. These nodes together correspond to the A_{22} of section 2.

Finally, in step 8 the $k \geq \alpha+1$ independent blocks created by removing the separators from the graph are numbered, using the well known reverse Cuthill-McKee algorithm [12].

Although the choice of α and the method of selection of the separators seems remarkably crude, we have found that attempts at more sophistication do not really yield significant benefits (except for some unrealistic, contrived examples). Just as in the regular rectangular grid case, the storage requirement, as a function of α, is very flat near its minimum. Even relatively large perturbations in the value of α, and in the selection of the separators, produced rather small changes in storage requirements.

§4 Some Implementation Details and Numerical Experiments

The linear equations solver used essentially regards the matrix as 2 by 2 partitioned, with the independent blocks formed by the removal of the separators forming the first partition, and the equations corresponding to the separators themselves comprising the second partition. Using the notation of (2.1) - (2.5), the linear equations solver stores L_{11} and L_{22} using a scheme similar to that proposed by Jennings [11], which exploits the variation in bandwidth. The matrix W_{12} is of course not stored; instead, the nonzeros of A_{12} are stored column by column in consecutive locations in a single one dimensional array, with a parallel array containing their row subscripts, and a pointer array indicating the position of the beginning of each column.

In order to gain some insight into the asymptotic behaviour of the ordering algorithm and the quality of the ordering produced, the ordering algorithm/solver combination was applied to problems derived from the graded L mesh shown in Figure 4.1, subdivided by increasing subdivision factors s, yielding s^2 as many triangles as in the original mesh. The numerical results are contained in Tables 4.1 and 4.2.

In Table 4.1, the column "storage" includes all array storage used by the program, including storage for pointers, auxiliary storage, space for the right hand side b, etc.. Note that the execution time of the ordering algorithm appears to grow linearly with N. The execution time associated with finding where the fill occurs, so that space for L_{11} and L_{22} can be allocated also appears to require time proportional to N. A description of this algorithm would take us too far afield; details can be found in [9]. Finally, note that the storage requirements grow as $N^{5/4}$, as the results of section 2 suggest.

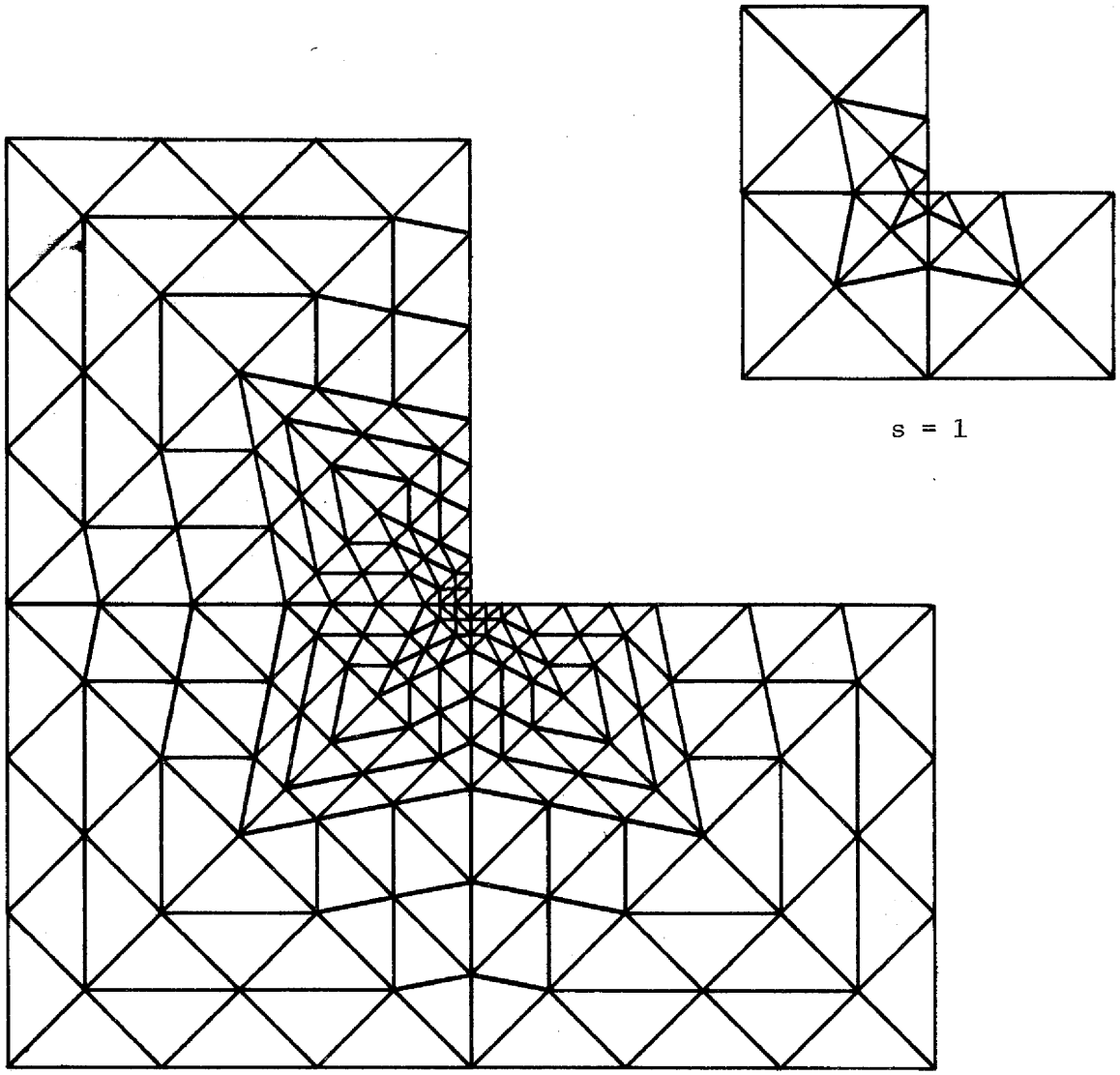

Figure 4.1 Graded L mesh.

In Tables 4.1 and 4.2 we distinguish between (1) execution times due to the determination of the ordering and the allocation of storage, (2) the time required to compute the factorization, and (3) the time required to compute x, given the factorization. In some situations many different problems having the same zero-nonzero structure must be solved, so it makes sense to ignore the cost of finding the ordering and setting up the data structures when evaluating the method. In other cases, many systems having the same coefficient matrix but different right hand sides must be solved. In this situation it may be sensible to evaluate the method on the basis of its solution time alone, and ignore ordering, allocation, and factorization times. All execution times reported are in seconds on an IBM 360/75. The programs are written in Fortran, and were compiled using the optimizing version of the compiler (OPT = 2).

The execution time of the factorization appears to be proportional to $N^{7/4}$, and the solution time appears to grow as $N^{5/4}$. These empirical results are not surprising in view of the results of section 2.

In order to demonstrate that such one-way dissection orderings do have a place in the "sophistication spectrum" of sparse matrix ordering schemes, we include in Tables 4.3 and 4.4 some further results on the graded L problem. The heading RCM refers to the results obtained using the reverse Cuthill-McKee ordering algorithm coupled with a standard band-oriented solver which exploits variation in the bandwidth. We regard this as the standard scheme. The heading NESTED represents the other end of the spectrum, and refers to results obtained by using a slightly revised version of the automatic nested dissection algorithm and solver described in [8].

The results of Table 4.3 are essentially self explanatory. Although asymptotically the nested dissection scheme requires less storage,

Table 4. 1 Performance statistics of the ordering algorithm, storage allocator, and storage requirements, for the graded - L problem and s = 4(1)12

s	N	Order Time	$\frac{\text{Order Time}}{N}$	Allocation Time	$\frac{\text{Alloc. Time}}{N}$	Storage	$\frac{\text{Storage}}{N^{5/4}}$
4	265	.18	6.79(-4)	.07	2.64(-4)	3486	3.26
5	406	.27	6.65(-4)	.11	2.71(-4)	5675	3.11
6	577	.38	6.58(-4)	.16	2.77(-4)	8581	3.03
7	778	.52	6.68(-4)	.21	2.69(-4)	12346	3.00
8	1009	.67	6.64(-4)	.27	2.67(-4)	16667	2.93
9	1270	.84	6.61(-4)	.35	2.76(-4)	21847	2.88
10	1561	1.03	6.60(-4)	.43	2.75(-4)	27860	2.83
11	1882	1.24	6.58(-4)	.52	2.76(-4)	34915	2.81
12	2233	1.47	6.58(-4)	.62	2.77(-4)	42636	2.77

Table 4.2 Performance statistics for the linear equations solver for the graded L problem, with s = 4(1)12.

s	N	Factorization Time	$\frac{\text{Fact. Time}}{N^{7/4}}$	Solution Time	$\frac{\text{Soln. Time}}{N^{5/4}}$
4	265	.85	4.88(-5)	.09	8.42(-5)
5	406	1.64	4.47(-5)	.14	7.68(-5)
6	577	2.89	4.25(-5)	.21	7.43(-5)
7	778	4.66	4.07(-5)	.29	7.06(-5)
8	1009	7.11	3.94(-5)	.40	7.03(-5)
9	1270	10.07	3.73(-5)	.51	6.73(-5)
10	1561	14.50	3.74(-5)	.65	6.62(-5)
11	1882	20.10	3.72(-5)	.80	6.45(-5)
12	2233	26.69	3.68(-5)	.98	6.38(-5)

Table 4.3 Ordering times for the RCM, one-way dissection and nested dissection algorithms, and the storage requirements for their respective solvers, for the graded-L problem.

s	N	ORDERING TIME			TOTAL STORAGE		
		RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED
4	265	.12	.18	.36	4279	3486	5433
5	406	.19	.27	.61	7764	5675	91350
6	577	.27	.38	.09	12748	8581	18845
7	778	.36	.52	1.27	19497	12346	19863
8	1009	.49	.67	1.72	28277	16667	26691
9	1270	.60	.84	2.25	39354	21847	35252
10	1561	.73	1.03	2.89	52994	27860	44957
11	1882	.88	1.24	3.54	69463	34915	55924
12	2233	1.04	1.47	4.35	89027	42638	68244

		FACTORIZATION						SOLUTION					
		OPERATIONS			TIME			OPERATIONS			TIME		
s	N	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED
4	265	.297	.473	.330	.34	.90	.69	.749	.651	.738	.06	.08	.11
5	406	.662	1.053	.685	.70	1.71	1.28	1.389	1.159	1.288	.12	.13	.18
6	577	1.288	1.808	1.201	1.27	2.99	2.05	2.318	1.749	1.994	.19	.20	.28
7	778	2.278	3.044	1.988	2.14	4.95	3.09	3.587	2.546	2.923	.29	.28	.38
8	1009	3.749	4.595	3.003	3.38	7.38	4.35	5.251	3.611	4.020	.41	.39	.52
9	1270	5.837	7.159	4.404	5.10	10.38	6.05	7.362	4.726	5.360	.58	.50	.68
10	1561	8.695	10.656	6.113	7.38	14.77	8.07	9.973	6.287	6.893	.77	.65	.87
11	1882	12.490	14.221	8.295	10.40	20.21	10.47	13.139	7.767	8.660	1.01	.80	1.07
12	2233	- ($\times 10^5$)	19.620 ($\times 10^5$)	10.084 ($\times 10^5$)	-	26.52	13.32	- ($\times 10^4$)	9.743 ($\times 10^4$)	10.631 ($\times 10^4$)	-	.98	1.30

Table 4.4 Operation counts and execution times for the linear equation solvers, for the three different orderings on the graded-L problem.

		FACTORIZATION						SOLUTION					
		OPERATIONS			TIME			OPERATIONS			TIME		
s	N	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED	RCM	ONE-WAY	NESTED
4	265	.297	.473	.330	.34	.90	.69	.749	.651	.738	.06	.08	.11
5	406	.662	1.053	.685	.70	1.71	1.28	1.389	1.159	1.288	.12	.13	.18
6	577	1.288	1.808	1.201	1.27	2.99	2.05	2.318	1.749	1.994	.19	.20	.28
7	778	2.278	3.044	1.988	2.14	4.95	3.09	3.587	2.546	2.923	.29	.28	.38
8	1009	3.749	4.595	3.003	3.38	7.38	4.35	5.251	3.611	4.020	.41	.39	.52
9	1270	5.837	7.159	4.404	5.10	10.38	6.05	7.362	4.726	5.360	.58	.50	.68
10	1561	8.695	10.656	6.113	7.38	14.77	8.07	9.973	6.287	6.893	.77	.65	.87
11	1882	12.490	14.221	8.295	10.40	20.21	10.47	13.139	7.767	8.660	1.01	.80	1.07
12	2233	- ($\times 10^5$)	19.620 ($\times 10^5$)	10.084 ($\times 10^5$)	-	26.52	13.32	- ($\times 10^4$)	9.743 ($\times 10^4$)	10.631 ($\times 10^4$)	-	.98	1.30

Table 4.4 Operation counts and execution times for the linear equation solvers, for the three different orderings on the graded-L problem.

because of differences in the coefficients of the estimates and differences in data structure complexity, the one-way scheme requires less storage until N is very large indeed. The one-way dissection ordering algorithm takes about fifty percent more time than the RCM algorithm, but it is considerably faster than the nested dissection algorithm.

Perhaps the most noteworthy aspect of Table 4.4 is that it illustrates how dangerous it is to conclude very much of a practical nature from a study of operation counts alone. While they correctly predict the trends in execution time, differences in data structure complexity can have an enormous effect on the operations-per-second output of a linear equations solver. Because the data structures are simplest for the standard scheme, it remains competitive in terms of execution time for values of N much larger than the operation counts would suggest.

There are a number of situations where the one-way dissection scheme could be quite attractive. Since its storage requirements are substantially lower than either of its competitiors over a fairly large range of N, in situations where storage is expensive and/or severely restricted, it may be the method of choice even though its execution time may be larger than its competitors. In addition, in some situations involving the solution of some mildly nonlinear or time dependent problems, many systems having the same coefficient matrix must be solved. In these situations, the cost of solving the problem, given the factorization, may be the primary factor governing the method's merit. The last six columns in Table 4.3 indicate that the one-way dissection scheme is a strong contender in these cases.

The one-way dissection approach has the potential to be very important in a multi-processor mini-computer environment. The computation

involving the leading α independent blocks can quite conveniently be performed on α different computers, in parallel.

§5 Concluding Remarks

We have presented a heuristic algorithm for finding a so-called one-way dissection ordering for an undirected graph. Although the experiments presented are for a single problem, quite extensive testing on a variety of other problems of the same type suggest that for these problems:

1. The ordering algorithm executes in $0(N)$ time.
2. The ordering produced, when used with the solver which exploits the structure of the reordered matrix as described in section 2, yields $0(N^{7/4})$ factorization times and $0(N^{5/4})$ storage requirements.

For fairly obvious reasons, the cross-over points where one scheme became more attractive than another (according to some specified criterion) depended upon the particular problem. For example, extremely long slender meshes yielded problems where the ordinary band scheme was uniformly superior for all s. However, for meshes which were undeniably two or three dimensional, there was a substantial range of N where the one-way scheme was, in some respects, the most efficient.

§6 References

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

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

[3] James R. Bunch and D.J. Rose, "Partitioning, tearing, and modification of sparse linear systems", J. Math. Anal. and Appl., 48 (1974), pp. 574-593.

[4] I.S. Duff, "Sparse matrices", AERE Rept HL 76/485, Harwell, England, February, 1976.

[5] Alan George, "On block elimination for sparse linear systems", SIAM J. Numer. Anal., 11 (1974), pp. 585-603.

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

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

[8] 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).

[9] 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, University of Waterloo, April, 1977.

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

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

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

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

[14] 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).

[15] Gilbert W. Strang and George J. Fix, An Analysis of the Finite Element Method, Prentice-Hall Inc., Englewood Cliffs, N.J. (1973).

[16] J.H. Wilkinson and C. Reinsch, Handbook for Automatic Computation II: Linear Algebra, Springer Verlag, 1971.

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