NUMERICAL EXPERIMENTS USING DISSECTION METHODS TO SOLVE n BY n GRID PROBLEMS

by

Alan George

Research Report CS-75-07

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

March 1975

This research was supported in part by Canadian National Research Council Grant A8111.

ABSTRACT

Recently the author has proposed two theoretically efficient orderings for Gaussian elimination when it is applied to systems of n^2 linear equations arising in connection with the use of finite elements methods on an n by n grid [6,7]. These are efficient in the sense that if zeros are exploited, the amount of arithmetic required is $O(n^3)$ or $O(n^{7/2})$, compared to $O(n^4)$ if the usual row by row numbering scheme is used. Similarly, the amount of fill suffered is $O(n^2 \log_2 n)$ and $O(n^{5/2})$ compared to $O(n^3)$. These comparisons ignored differences in the program and data structure complexity required to exploit the zeros for the different orderings. In this paper the author describes how these orderings can be implemented in an efficient manner and provides numerical experiments which show that the performance of these programs properly reflects the theory.

§1 Introduction

Recently the author has proposed two efficient ordering strategies for sparse positive definite systems of linear equations associated with an n by n grid or mesh consisting of $(n-1)^2$ small "elements" [6,7]. For some numbering of the grid points from 1 to n^2 we have the n^2 by n^2 matrix equation

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

where $A_{ij} = 0$ unless grid points i and j belong to the same element. Matrix problems such as this or similar to this arise in connection with the use of finite element or finite difference methods to solve elliptic boundary value problems on rectangular domains. We solve (1.1) using symmetric Gaussian elimination or Cholesky's method by first factoring A into LL^T, where L is lower triangular, and then solving the triangular systems $Ly = b$ and $L^Tx = y$.

Now it is well known that when Gaussian elimination is applied to a sparse matrix, it usually suffers some "fill"; that is, the triangular factors have nonzero components in positions which are zero in the original matrix. Thus, we might reduce this fill by replacing (1.1) above by the equivalent problem

$$(PAP^T)q = pb, \tag{1.2}$$

where $q = Px$ and P is a judiciously chosen permutation matrix of the appropriate size. The permutation P might be chosen for other reasons, such as reduced arithmetic requirements, convenient data management, convenient coding and perhaps other factors.

The two proposed ordering strategies mentioned above (and described in sections 3 and 4) are efficient in the sense that the amount of arithmetic required to solve the problems is $O(n^3)$ and $O(n^{7/2})$, compared to $O(n^4)$ for the so-called row by row or "natural" ordering. Similarly, the fill suffered during the factorization is $O(n^2 \log_2 n)$ and $O(n^{5/2})$, compared to $O(n^3)$ for the natural ordering.

From a pure complexity point of view, these comparisons are perefectly legitimate. However, from a practical point of view their relevancy is open to question because they ignore differences in the cost of exploiting sparsity for the different orderings, and this cost appears to be higher for the more efficient orderings. Also, the constants multiplying the high order terms in the arithmetic and fill counts are different for each ordering.

This motivates the objective of this paper, which is to compare in a practical way (i.e., computer execution time, computer storage requirements), the two proposed ordering strategies and the natural (row-by-row) ordering.

The paper consists of the following. In sections 2-4 we review the natural, nested dissection, and one-way dissection orderings respectively. We include estimates for the number of nonzero components in their respective L's that we must store, and the amount of arithmetic required to factor the $n^2 \times n^2$ matrix, for each ordering. We also describe the storage techniques used and provide estimates in each case for the amount of storage overhead (pointer data etc.) required for the data structure. Finally, in these

sections we report the results of some experiments which demonstrate that our implementations properly reflect the theory. In section 5 we provide a comparison among the codes, with respect to various practical criteria. Section 6 contains our concluding remarks.

It is helpful to have some notation for various quantities we use to illustrate and compare the performance and efficiency of our codes. In this connection, we provide the following, where "multiplications" mean multiplicative operations (multiplications and divisions).

θ_F - the number of multiplications required to compute the Cholesky factorization, if all zeros are exploited.

$\bar{\theta}_F$ - the number of multiplications actually performed by the code under discussion, to compute the Cholesky factorization. (Obviously $\theta_F \leq \bar{\theta}_F$.)

θ_S - the number of multiplications required to solve $Ax = b$, given the factorization produced by the code under study, if all zeros are exploited. (We assume that b is full.)

$\bar{\theta}_S$ - the number of multiplications actually performed in solving $Ax = b$, given the factorization produced by the code under study. Again, it is clear that $\theta_S \leq \bar{\theta}_S$. (All our codes assume b is full.)

η_L - the number of nonzero components in L that we need to store in order to execute the algorithm under study. (For the one-way dissection scheme, this is not the number of nonzero components in L. See section 3 for details.)

$\bar{\eta}_L$ - the number of components of L actually stored, including any zeros.

$\bar{\eta}_L^0$ - the number of storage locations used for pointers etc. in connection with storing L.

$\bar{\bar{\eta}}_L$ - the total storage used in connection with the storage of L ($= \bar{\eta}_L + \bar{\eta}_L^0$).

η - the total number of storage locations used by the program for its data, including that required for the right-hand side b, all pointer data, and any auxiliary vectors that are needed to execute the algorithm.

T_F - central processor time in seconds required to compute the factorization of A.

T_S - central processor time required to solve Ax = b, given the factorization of A.

We believe the times T_F and T_S to be in error by no more than about 5 percent.

Our programs do not do any "packing" of pointers; that is, our codes used the same number of bits to store pointers as that used for floating point data, permutation vectors etc. However, on some computers it is convenient and desirable to store pointers using only a fraction of the storage required for a floating point number. Thus, the reader should keep in mind that some of our reported quantities such as $\bar{\eta}_L^0$ may be much smaller for similar implementations on other computers.

§2 Banded Orderings and their Implementations

In this and the following two sections we describe the three orderings which we intend to compare along with the programming techniques used in order to utilize them. The first is the simple but popular and widely used row by row numbering scheme, which yields a band matrix having bandwidth $m(A) = n + 1$, where

$$(2.1) \qquad m(A) = \max_{A_{ij} \neq 0} |i-j|.$$

Implicit in the use of band orderings is the assumption that zeros outside the band are to be exploited. The fill is confined within the band, and it is usual to ignore zeros within it when processing A. A storage scheme which is often used in conjunction with symmetric $N \times N$ band matrices is to store the diagonal and the m non-null sub-diagonals of A in an $N \times (m+1)$ rectangular array [14]. This storage scheme is reasonably efficient if $m << N$, and since no pivoting is required to maintain numerical stability, the factorization can be conveniently carried out "in place", with L replacing A in the array.

A somewhat more attractive scheme, in our opinion, is that suggested by Jennings [13]. Define $f_i = \min\{j | A_{ij} \neq 0\}$, $1 \leq i \leq N$. We then define the envelope of A by

$$(2.2) \qquad Env(A) = \{(i,j) | i \geq f_j \text{ and } j \geq f_i\}.$$

Jennings proposes storing those components a_{ij} of the lower triangle of A, for which $(i,j) \in Env(A)$, row by row in a one dimensional array S, with an extra N pointers to the diagonal components a_{ii} in S. Thus, this

storage scheme exploits any variation in the band, and avoids the inevitable wastage of m(m-1)/2 storage locations which the diagonal storage scheme described above incurs. (See [9 , chapter 4] for experiments which demonstrate how substantial these savings can be for some rectangular problems.) In exchange, we must store N pointers. We have used this storage scheme in the experiments we report here and in section 5 for the row by row numbering scheme. See Figure 2.1 for an example.

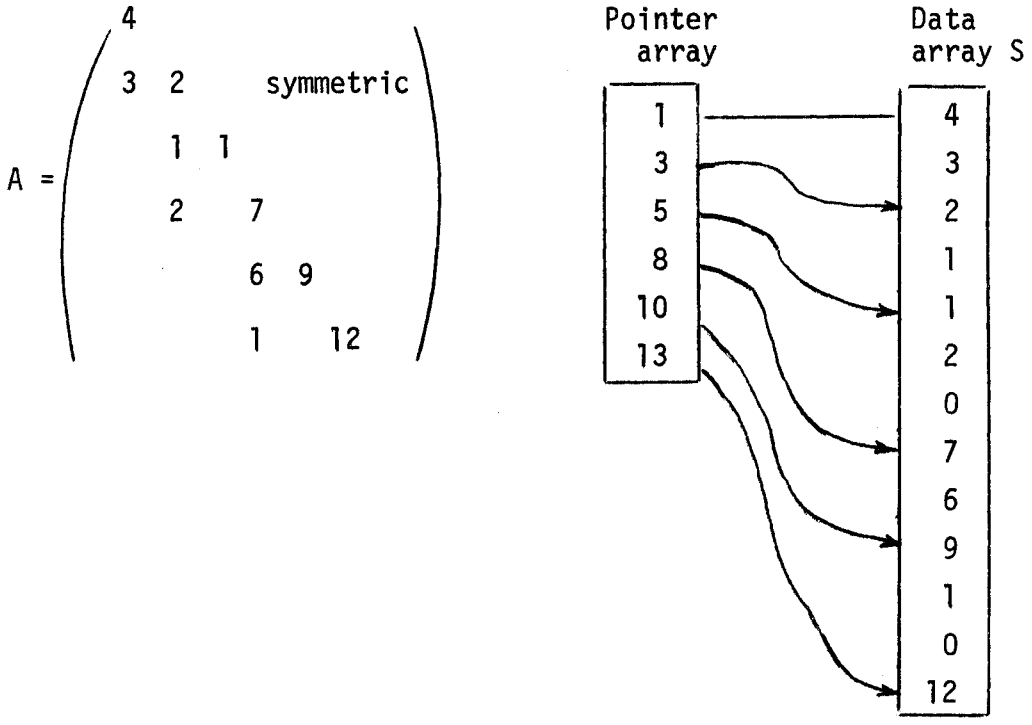

Fig.2.1 Illustration of Jennings envelope storage scheme

For the problems we consider in this paper, the number of nonzero components in L (η_L) is $n^3 + n^2 - n$. For our code we observed that $\eta_L = \bar{\eta}_L$ and $\bar{\bar{\eta}}_L = \eta_L + n^2$ as expected. It is easy to verify that $\theta_F \doteq n^4/2 + 11n^3/6 + O(n^2)$,

and our code satisfies $\bar{\theta}_F = \theta_F$. Obviously, $\theta_S = 2\eta_L$, and for our code we found that $\bar{\theta}_S = \theta_S$. It is also clear that $\eta_L^0 = n^2$ for our test problems.

To verify that our code's execution times T_F and T_S are proportional to $\bar{\theta}_F$ and $\bar{\theta}_S$ respectively, we present Table 2.1 which shows that the execution time properly reflects the amount of arithmetic being performed.

n	n^2	$T_F/\bar{\theta}_F$	$T_S/\bar{\theta}_S$
10	100	2.2(-5)	1.3(-5)
15	225	1.8(-5)	1.2(-5)
20	400	1.7(-5)	1.2(-5)
25	625	1.7(-5)	1.2(-5)
30	900	1.6(-5)	1.2(-5)
35	1225	1.6(-5)	1.2(-5)
40	1600	1.6(-5)	1.2(-5)

Table 2.1 Ratio of execution time to operation counts using the natural ordering and Jennings envelope storage scheme.

Table 2.1 demonstrates the interesting fact that the IBM optimizing compiler does a considerably better job of generating code for one of the inner loops of the triangular solvers than it did for the inner loop of the factorization routine. This is particularly surprising since they do not look all that different, as shown below.

```
      DO 4 J4 = J1, J2
           S  = S - A(J4)*A(J3)
           J3 = J3 + 1
4     CONTINUE
```

Inner loop of the factorization subroutine.

```
      DØ 4 K = K1, K2
           S = S - A(K)*A(L)
           L = L + 1
4     CONTINUE
      Inner loop of the lower triangular solver.
      DØ 5 J = J1, J2
        X(J) = X(J) - A(M)*R
        M    = M + 1
5     CONTINUE
      Inner loop of the upper triangular solver.
```

As expected, the machine code produced for the first two loop was the same. However, the code for the inner loop of the upper triangular solver was considerably more efficient.

This phenomenon, while not at all uncommon, serves to emphasize the danger in drawing conclusions about performance of algorithms without actually implementing them. Although this particular anomoly could be dismissed as "compiler dependent", many algorithms, while equivalent in some theoretical sense, differ considerably in their complexity of implementation.

§3 Nested Dissection Orderings and their Implementation

It is becoming increasingly well known that if we avoid operating on and storing all zeros, orderings which minimize m(A) or Env(A) are often far from optimal in the least-fill or least-arithmetic sense. The second ordering strategy we consider is the so-called nested dissection ordering scheme, which minimizes (in the asymptotic sense) both arithmetic and fill. For a detailed discussion, the reader is referred to George [6]. However, for completeness and to help describe the implementation, we review the basic idea here

Following the approach described in [1], let V be the set of vertices of the n by n mesh and let C_1 consist of the vertices on a mesh line which as nearly as possible divides V into two equal components R_1^1 and R_1^2. By definition, variables associated with vertices in R_1^1 are not connected to those in R_1^2. Thus, if we number variables in R_1^1, followed by those in R_1^2, followed finally by those in C_1, we induce the following block structure in A.

$$(3.1) \qquad A = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{13}^T & A_{23}^T & A_{33} \end{pmatrix}$$

Thus, variables associated with C_1 "dissect" the problem into two independent components. Obviously,

$$V \setminus C_1 = R_1^1 \cup R_1^2 = R_1.$$

Now choose vertex sets $S_1^\ell \subset R_1^\ell$, $\ell = 1,2$, consisting of nodes lying on mesh lines which as nearly as possible divide R_1^ℓ into equal parts. Let

$S_1 = S_1^1 \cup S_1^2$ and define $C_2 = C_1 \cup S_1$. If we number the variables associated with the vertices in $R_1^\ell \setminus S_1^\ell$ before those associated with S_1^ℓ, $\ell = 1,2$, then we induce in A_{11} and A_{22} of (3.1) exactly the same structure as that of the overall matrix, but of course on a smaller scale. Obviously, we can repeat this process, generating a nested sequence of vertex sets

$$\phi \subset C_1 \subset C_2 \subset \ldots \subset C_r = V, \quad \text{where}$$

$$V \setminus C_k = R_k = \bigcup_{\ell=1}^{2^k} R_k^\ell \qquad \text{and}$$

$$C_{k+1} \setminus C_k = S_k = \bigcup_{\ell=1}^{2^k} S_k^\ell.$$

An example of such a nested dissection appears in Figure 3.1 for a 10 by 10 mesh.† The vertex label k denotes membership in S_k.

Thus, an ordering induced by this nested dissection is as follows. Number variables associated with vertices in S_r^ℓ consecutively, followed by those in S_{r-1}^ℓ, and so on, finally numbering variables associated with vertices in S_1^1, S_1^2 and S_0 (= C_1). Each S_k^ℓ consists of part of a mesh line of nodes; for reasons which will be apparent later, consecutive numbers should be assigned to adjacent nodes, beginning at one end. Figure 3.2 contains an example of a numbering induced by the dissection of Figure 3.1.

---

† When there is a choice, we choose $|S_k^\ell|$ to be as small as possible, which means for our test problems the R_k^ℓ consist of rectangular vertex sets whose lengths do not exceed twice their width.

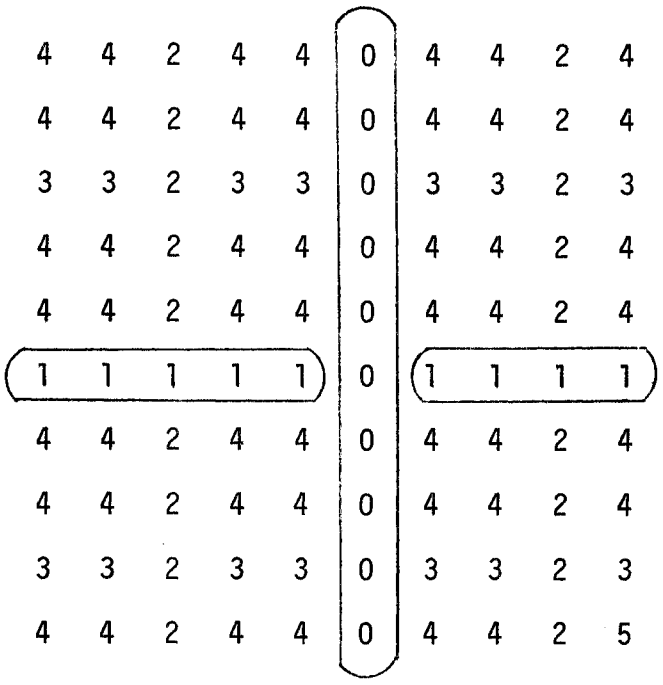

Fig.3.1 Nested dissection labelling of 10 by 10 mesh, with label k indicating membership in S_k. Sets S_0 and S_1^i, i = 1,2, are circled.

49	48	81	35	34	100	21	20	72	7
47	46	80	33	32	99	19	18	71	6
63	62	79	61	60	98	59	58	70	57
45	44	78	31	30	97	17	16	69	5
43	42	77	29	28	96	15	14	68	4
90	89	88	87	86	95	85	84	83	82
41	40	76	27	26	94	13	12	67	3
39	38	75	25	24	93	11	10	66	2
56	55	74	54	53	92	52	51	65	50
37	36	73	23	22	91	9	8	64	1

Fig.3.2 An ordering induced by the labelling of Fig.3.1.

We now describe the implementation scheme. It is helpful in this connection to denote the set of nodes which are adjacent to R_k^ℓ, but not in R_k^ℓ, by ∂R_k^ℓ. We also denote the node numbers assigned to the nodes in the set Q by X(Q).

Our strategy is to regard A as a partitioned matrix, with the partitioning determined by the sets S_k^ℓ. This determines "block columns" of L, and it turns out that the rows in these block columns themselves can be partitioned so as to have $t \leq 4$ non-null blocks. To see this, consider the following:

1) The set ∂R_k^ℓ "insulates" the nodes in R_k^ℓ from the rest of V. In terms of Gaussian elimination, this means that the elimination of variables of R_k^ℓ can only connect variables of $R_k^\ell \cup \partial R_k^\ell$. (For example, in (3.1), the first part of the factorization involving A_{11} has no effect on A_{22}; that is $\partial R_1^1 (= C_1)$ insulates R_1^1 from the remainder of V.) Thus, if $p \in X(R_k^\ell)$, $L_{qp} = 0$ unless $q \in X(R_k^\ell \cup \partial R_k^\ell)$.

2) If $p \in X(R_k^\ell \setminus S_k^\ell)$, $q \in X(S_k^\ell)$ and $r \in X(\partial R_k^\ell)$, then $p < q < r$. Thus, in terms of L, the only rows which are involved in the block column of L corresponding to S_k^ℓ are those with numbers in $X(S_k^\ell \cup \partial R_k^\ell)$.

3) ∂R_k^ℓ consists of portions of mesh lines, and these portions have consecutive node numbers. Thus, $X(\partial R_k^\ell) = \bigcup_{m=1}^{t} X(\partial R_k^{\ell,m})$, $t \leq 4$, where $X(\partial R_k^{\ell,m})$ consists of a sequence of consecutive integers. Also, nodes of S_k^ℓ are numbered consecutively.

Observations 1) and 2) imply that for $p \in X(S_k^\ell)$, $L_{jp} = 0$ unless $j \in X(S_k^\ell \cup \partial R_k^\ell)$. Observation 3) implies that the block column of L corresponding to S_k^ℓ can be partitioned so that $L_{jp} \neq 0 \Rightarrow j$ is in one of 5 contiguous subsets of $\{p,p+1,\ldots,n\}$. Specifically, the components L_{jp} to be stored have subscripts (j,p), $j \geq p$, where $p \in X(S_k^\ell)$ and j is in $X(S_k^\ell)$ or one of $X(\partial R_k^{\ell,m})$, $1 \leq m \leq t$. Although it would take us too far afield here, it is easy to see, using the generalized element model developed in [6], that the non-null blocks of L will be full or nearly so.

In Fig.3.2 the members of a typical R_k^ℓ are circled, with the members of ∂R_k^ℓ put in squares. Fig.3.3 contains L corresponding to the ordering of Fig.3.2 with the partitioning of L indicated by lines drawn on it.

The number of diagonal blocks n_d (i.e., the number of sets S_ℓ^k) satisfies $n_d < n^2/2 + n$. Each block column of L has at most 4 off-diagonal blocks. Moreover, each of these blocks has the same number of columns, and the diagonal block is square. Thus, we need $5n^2/2 + 0(n)$ numbers to record the dimensions of the blocks. In our implementation, the diagonal blocks were stored using Jenning's scheme, thus requiring a further n^2 pointers, and the off-diagonal blocks were all allocated from a single one dimensional array, so we needed a further $2n^2 + 0(n)$ pointers to record their position in the array. We also need $5n^2/2 + 0(n)$ integers indicating the row number of L which corresponds to the first row of each of the blocks. Finally, for convenience in coding, it was helpful to have an array q where q_i, $1 \leq i \leq n^2$, indicates the block column in which column i resides. Thus, for this storage scheme and the nested dissection ordering, an estimate for $\bar{\eta}_L^0$ is given by

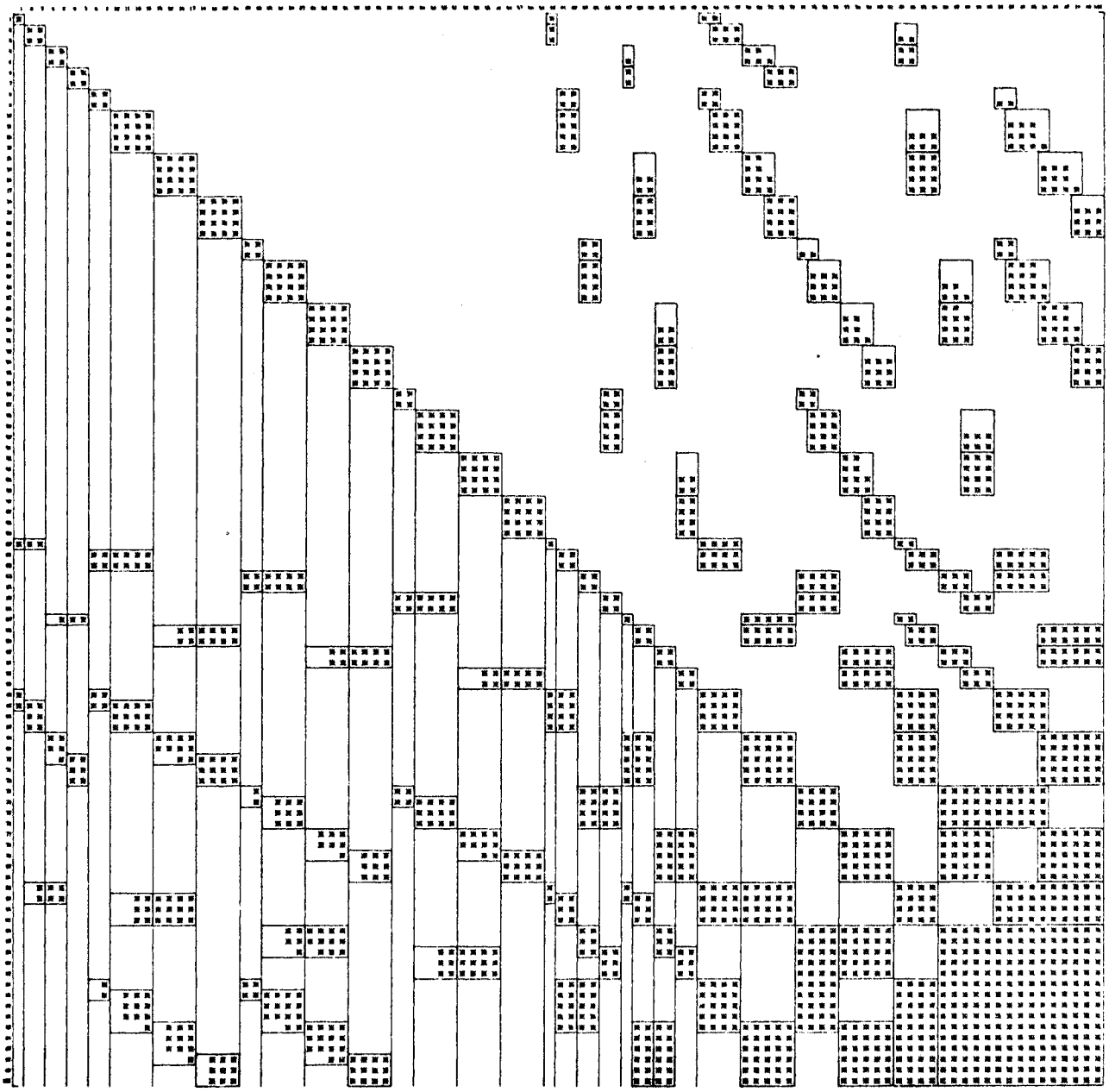

Fig.3.3 The matrix structure of $L+L^T$ corresponding to the ordering of Fig.3.2. The block columns determined by the S_k^{ℓ} are indicated; the enclosed blocks on the diagonal and above the diagonal are the matrix components actually stored.

$$(3.2) \qquad \bar{\eta}_L^0 = 9n^2 + O(n).$$

Using the techniques developed by Birkhoff and George [1], the following estimates for θ_F and η_L can be derived [10].

$$(3.3) \qquad \theta_F^e = 9.88n^3 - 17n^2\log_2(n+1) + 16.06n^2 - 0.50n\log_2(n+1) - 24.69n + 11.50\log_2(n+1).$$

$$(3.4) \qquad \eta_L^e = 7.75n^2\log_2(n+1) - 24n^2\log_2(n+1) + 11.50n\log_2(n+1) + 11n + .75\log_2(n+1).$$

The superscript e is intended to remind the reader that (3.3) and (3.4) are estimates.

Before describing the performance of our nested dissection code, we want to establish that the orderings we are using do indeed yield operation counts and numbers of nonzero components given approximately by (3.3) and (3.4). Table (3.1) contains the ratios $|\theta_F-\theta_F^e|/\theta_F$ and $|\eta_L-\eta_L^e|/\eta_L$ for the orderings used by our code. These ratios show to our satisfaction that the orderings we use yield values of θ_F and η_L which are well-approximated by (3.3) and (3.4).

n	n^2	$\lvert\theta_F-\theta_F^e\rvert/\theta_F$	$\lvert\eta_L-\eta_L^e\rvert/\eta_L$
10	100	.215	.054
15	225	.005	.065
20	400	.017	.017
25	625	.007	.020
30	900	.018	.013
35	1225	.008	.008
40	1600	.0004	.003

Table 3.1 Relative error in the estimates θ_F^e and η_L^e provided by (3.3) and (3.4)

Just as we did for the envelope code of section 2, we now want to show that the execution times T_F and T_S properly reflect the amount of arithmetic being performed. Table 3.2 contains the ratios $T_F/\bar{\theta}_F$ and $T_S/\bar{\theta}_S$ for n = 10(5)40.

n	n^2	$T_F/\bar{\theta}_F$	$T_S/\bar{\theta}_S$
10	100	2.1(-5)	2.3(-5)
15	225	2.5(-5)	3.1(-5)
20	400	1.7(-5)	1.8(-5)
25	625	1.6(-5)	1.6(-5)
30	900	1.7(-5)	2.1(-5)
35	1225	1.5(-5)	1.8(-5)
40	1600	1.4(-5)	1.6(-5)

Table 3.2 Ratios $T_F/\bar{\theta}_F$ and $T_S/\bar{\theta}_S$ for the nested dissection code

Note first that in Table 3.2, for $n \geq 20$, it appears that the nested dissection code is at least as efficient as the band (envelope) code discussed in §2 (see Table 2.1). Thus, we conclude that even for relatively small problems our more elaborate data structure does not seriously penalize the operations-per-second output of our code. The somewhat erratic nature of the ratios in Table 3.2 will be explained after we consider the efficiency of our storage scheme, which we turn to now.

In a sense, the efficiency of our storage scheme has already been established. Using our estimates (3.2) and (3.4) for $\bar{\eta}_L^0$ and η_L, we see that $\bar{\eta}_L^0/\eta_L \to 0$ as $n \to \infty$, which is an enviable situation. By comparison, many sparse matrix storage schemes require one or more pointers for each nonzero

component of L stored [12,15]. The only item to confirm is that $\bar{\eta}_L - \eta_L$ is acceptably small; that is, that we do not store many zeros. Table 3.3 below contains the ratios $\eta_L/\bar{\eta}_L$ and $\bar{\eta}_L^0/\bar{\eta}_L$ for n = 10(5)40.

n	n^2	$\eta_L/\bar{\eta}_L$	$\bar{\eta}_L^0/\bar{\eta}_L$	$\bar{\bar{\eta}}_L$
10	100	.93	.573	1705
15	225	1.0	.797	4993
20	400	.954	.408	8845
25	625	.940	.333	14775
30	900	.990	.474	24481
35	1225	.986	.396	33707
40	1600	.967	.298	45007

Table 3.3 Ratio of nonzero components in L to the number actually stored, the ratio of overhead storage to the number of components of L stored, and the total storage $\bar{\bar{\eta}}_L$ required for L (= $\bar{\eta}_L + \bar{\eta}_L^0$)

From the ratio $\eta_L/\bar{\eta}_L$ in Table 3.3, it seems fair to conclude that our data structure is appropriate in the sense that relatively few zero components of L are being stored. Also, the ratio $\bar{\eta}_L^0/\bar{\eta}_L$ is tending slowly to zero as expected. However, just as in Table 3.2, the ratios behave somewhat erratically, despite their overall trend. We now offer an explanation of this behaviour.

Unless $n = 2^r - 1$, r an integer, the dissection will be "imperfect" in the sense that some or all of the sets S_k^ℓ at the final level of dissection will contain more than a single vertex (as was the case in the example described by Figures 3.1-3.3). This has two consequences. First, some of

the blocks will contain some zeros, implying that $\eta_L/\bar{\eta}_L < 1$. The variation of this ratio in Table 3.2 illustrates this phenomenon. Second, for $n \neq 2^{\ell}-1$, n_d will usually be considerably smaller than the bound $n^2/2+n$. Since a major component of $\bar{\eta}_L^0$ and a major portion of the computation overhead is directly proportional to n_d, a relatively imperfect dissection is beneficial with regard to reducing $\bar{\eta}_L^0$ and execution overhead. This explains the variation in Table 3.2 and in the second column of Table 3.3.

This latter observation led us to speculate whether it might be possible to decrease $\bar{\bar{\eta}}$ and/or execution time by "suspending" the dissection "early"; that is, to simply number the vertices in R_k^{ℓ} arbitrarily as soon as $|R_{\ell}^k|$ fell below a certain threshold. Since $\bar{\bar{\eta}}_L = \bar{\eta}_L^0 + \bar{\eta}_L$, the storage question was whether early suspension of the dissection might decrease η_L^0 more than it increased $\bar{\eta}_L$. Our experience was that in the relatively few cases where the idea did decrease $\bar{\bar{\eta}}_L$ and/or execution time, the decrease was not significant, so we rejected the idea. However, on other computers, the idea might very well be effective.

§4 One-Way Dissection Schemes and their Implementation

Our final scheme under study, which we shall refer to as one-way dissection, may be regarded as a compromise between band and nested dissection orderings. The basic idea is to dissect the mesh in such a way that the independent blocks can be efficiently processed by band or envelope methods. Theoretically, this schemes' efficiency lies between the band and nested dissection orderings, since it has been shown that the computation and storage requirements are proportional to $n^{7/2}$ and $n^{5/2}$ respectively. Following [7], the scheme is as follows.

Let α be an integer satisfying $1 < \alpha << n$, and choose α-1 sets of grid lines (separators) which dissect the n by n mesh into α independent blocks, as depicted in Figure 4.1 where $\alpha = 4$. The heavy lines are intended to depict a grid line, and the circled numbers indicate the order in which vertex sets are to be labelled.

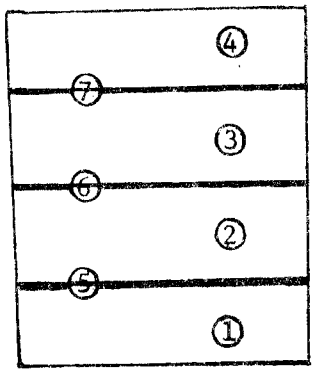

Fig.4.1 One-way dissection of an n by n mesh with $\alpha = 4$

Now number the α independent blocks column by column, followed by the α-1 separators in any order. For our example, such an ordering induces the block structure shown in Figure 4.2. The hatched blocks indicate the blocks

which are created when the factorization is carried out. The pattern for general α should be clear.

There are two key observations which are essential to the use of this ordering scheme. We illustrate the ideas with a block 2 by 2 symmetric positive definite matrix A below, where A_{12} is assumed to be non-null.

$$(4.1) \qquad A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} .$$

First, given A, we can compute either of the factorizations shown below.

$$(4.2) \qquad \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} = \begin{pmatrix} L_1 & 0 \\ W^T & L_2 \end{pmatrix} \begin{pmatrix} L_1^T & W \\ & L_2^T \end{pmatrix}$$

$$(4.3) \qquad \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & 0 \\ A_{12}^T & \tilde{A}_{22} \end{pmatrix} \begin{pmatrix} I & \tilde{W} \\ 0 & I \end{pmatrix}$$

Here $A_{11} = L_1 L_1^T$ and $L_2 L_2^T = A_{22} - A_{12}^T A_{11}^{-1} A_{12} = \tilde{A}_{22}$, where L_1 and L_2 are lower triangular. The off-diagonal blocks of the factorizations are defined by

$$(4.4) \qquad W = L_1^{-1} A_{12},$$

and

$$(4.5) \qquad \tilde{W} = L_1^{-T} W = A_{11}^{-1} A_{12}.$$

Note the factorization (4.3) is as useful as (4.2) since we compute and store L_1 and L_2 rather than retaining A_{11} and $\tilde{A}_{22}$. The essential difference in computing the factorizations is whether $\tilde{A}_{22}$ is computed as

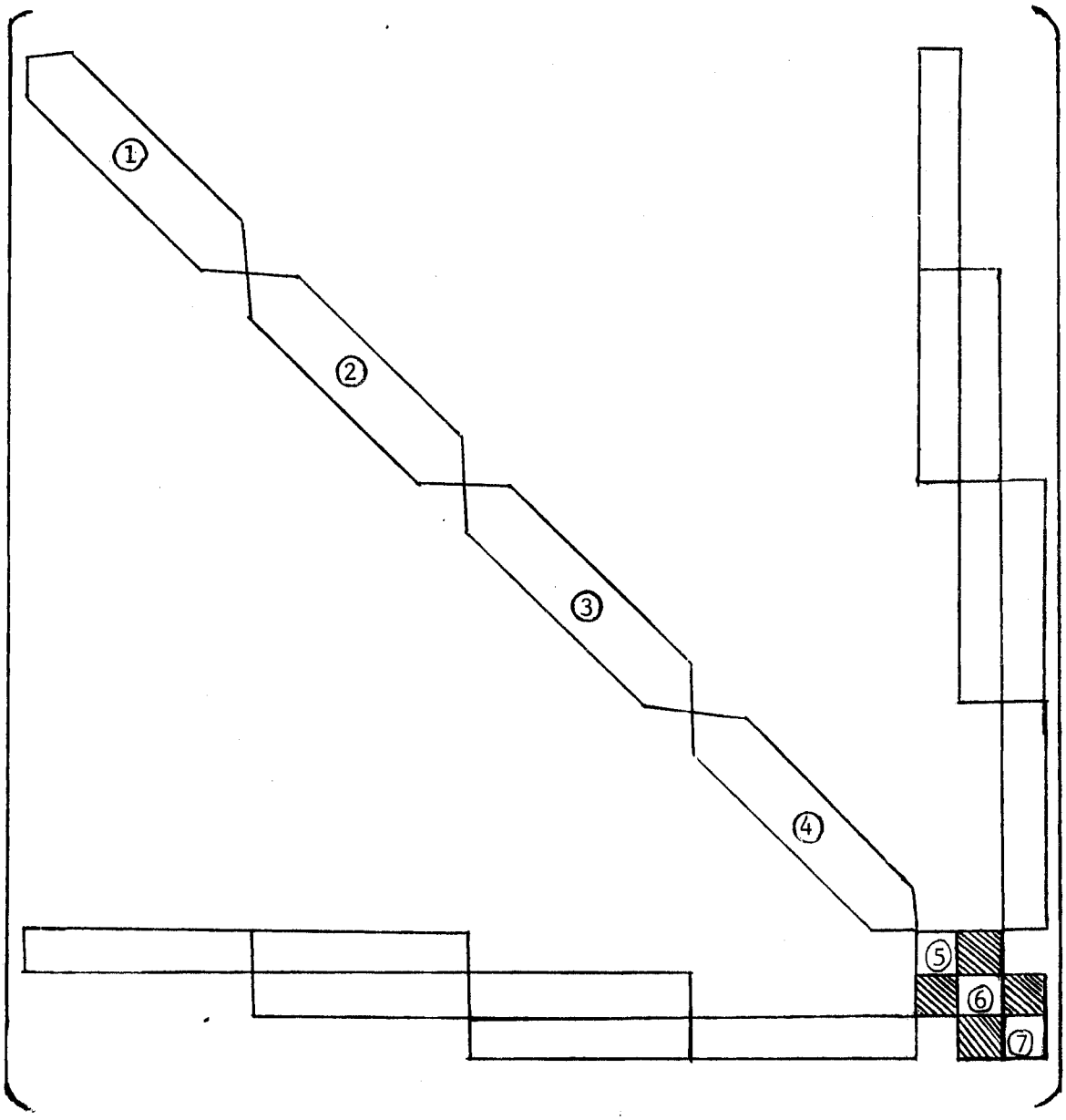

Fig.4.2 Block structure in A induced by one-way dissection with $\alpha = 4$

$A_{22} - (A_{12}^T L_1^{-T})(L_1^{-1} A_{12})$ or $A_{22} - A_{12}^T (L_1^{-T}(L_1^{-1} A_{12}))$. As shown by George [8], these factorizations in general require different amounts of arithmetic to compute, and the unsymmetric factorization (4.9) may be cheaper to compute than (4.8), even though A is symmetric!

The second key observation is that we may not wish to retain the off-diagonal blocks of the factorization. Bunch and Rose [3] observe that in performing the solution, given the triangular factorization (4.8), it may require fewer arithmetic operations to compute $\tilde{x}_1 = Wx_2$ by computing $\bar{x}_1 = A_{12}x_2$ and then solving $L_1\tilde{x}_1 = \bar{x}_1$, than simply multiplying x_2 by W. Similar remarks apply to the use of W^T, W and $\tilde{W}^T$.

Returning to our one-way dissection ordering, it turns out to be beneficial to use both ideas. Indeed, if we do not, the number of multiplications required to factor the matrix using this ordering is $>> \frac{1}{2}n^4$, and the number of nonzero components in L is $>> n^3$. That is, the ordering is much inferior to the natural ordering.

However, our strategy is to let A_{11} of (4.1) correspond to the α independent blocks, and A_{22} correspond to the remaining unknowns. Thus, for example, in Fig.4.2, the only part of L that is stored is the lower triangle of the diagonal blocks and the hatched blocks in the lower triangle.

Our implementation is straightforward. The matrices A_{11} and $\tilde{A}_{22}$ are stored using Jennings envelope scheme. This incurs a storage overhead of n^2 pointers. Since A_{12} is very sparse, only its nonzero components are stored. It is easy to verify that it has at most 6 nonzero components in each column. Two N_1 by 6 arrays are used, (where $N_1 = n(\alpha-1)$,) one

containing the nonzero components of A_{12}^T and the other containing their respective column subscripts. Thus, the overhead storage $\bar{\eta}_L^0$ required for this storage scheme is

$$\bar{\eta}_L^0 = n^2 + 6n(\alpha-1). \tag{4.6}$$

It is straightforward although tedious to show that the number of nonzero components of L that we must store, including those of A_{12} and including $\bar{\eta}_L^0$, is given approximately by

$$\bar{\eta}_L^e = \frac{\alpha}{2}(3n^2+23n-6) + \frac{n}{\alpha}(n^2+n-2) - \frac{n}{2}(3n+17). \tag{4.7}$$

The number of multiplications required to factor the matrix, as a function of α, is given approximately by

$$\theta_F^e(\alpha) = \frac{\alpha}{6}(7n^3+27n^2-40n-26) + \frac{1}{6\alpha}(18n^4+24n^3-81n^2-90n) - \frac{1}{6\alpha^2}(15n^4+13n^3-15n^2-9n) - \frac{1}{6}(13n^3-12n^2-136n), \tag{4.8}$$

and the number of multiplications required to solve the system, given the factorization, is given approximately by

$$\theta_S^e(\alpha) = 3\alpha(n^2+3n-4) + \frac{4}{\alpha}(n^3+n^2-n) - 5n^2 + 3n. \tag{4.9}$$

For large n, (4.7), (4.8) and (4.9) are approximately minimized respectively by

$$\alpha_\eta^e = \sqrt{2n/3}, \tag{4.10}$$

$$\alpha_F^e = \sqrt{18n/7},$$

and

(4.12) $\alpha_S^e = \sqrt{4n/3}$.

Just as we did with our nested dissection code, we wish to investigate the performance of our one-way dissection code to see how well its behaviour is predicted by the theory just developed. It turns out to make no sense to distinguish between θ_F and $\bar{\theta}_F$, or between θ_S and $\bar{\theta}_S$, since no zeros are operated upon. Moreover, it is easy to show that $|\bar{\eta}_L - \eta_L| = 4(\alpha-1)$, which is relatively insignificant. Thus, if our program properly reflects our theory, the quantities $\theta_F^e(\alpha)$, $\theta_S^e(\alpha)$, and $\eta_L^e(\alpha)$ should provide reasonable estimates for $\bar{\theta}_F(\alpha)$, $\bar{\theta}_S(\alpha)$ and $\bar{\bar{\eta}}_L$. Table 4.1 shows this to be the case for n = 40. Similar results hold for other values of n.

α	$\lvert\bar{\theta}_F(\alpha)-\theta_F^e(\alpha)\rvert/\bar{\theta}_F(\alpha)$	$\lvert\bar{\theta}_S(\alpha)-\theta_S^e(\alpha)\rvert/\bar{\theta}_S(\alpha)$	$\lvert\bar{\bar{\eta}}_L(\alpha)-\bar{\bar{\eta}}_L^e(\alpha)\rvert/\bar{\bar{\eta}}_L(\alpha)$
2	.016	.007	.004
3	.023	.009	.006
4	.007	.008	.008
5	.014	.006	.010
6	.009	.004	.011
7	.003	.001	.012
8	.031	.002	.013
9	.015	.002	.011
10	.039	.008	.013
11	.021	.007	.012
12	.030	.008	.012

Table 4.1 Relative error in the estimates $\theta_F^e(\alpha)$, $\theta_S^e(\alpha)$ and $\bar{\bar{\eta}}_L^e(\alpha)$ for n = 40 and α = 2(1)12.

We were interested in determining the reliability of the values of α predicted by (4.10)-(4.12). To this end we present Table 4.2 below.

n	n^2	α_η^e	$\bar{\alpha}_\eta$	α_F^e	$\bar{\alpha}_F$	α_F^T	α_S^e	$\bar{\alpha}_S$	α_S^T
10	100	2.58	2	5.07	3	2	3.65	3	3
15	225	3.16	3	6.21	5	5	4.47	4	4
20	400	3.65	3	7.17	5	5	5.16	5	5
25	625	4.08	4	8.02	6	5	5.77	6	5
30	900	4.47	4	8.78	7	6	6.32	6	6
35	1225	4.83	4	9.49	9	7	6.83	7	6
40	1600	5.16	5	10.14	10	8	7.30	7	7

Table 4.2 Predicted optimal values of α given by (4.10)-(4.12). The values of $\bar{\alpha}_\eta$, $\bar{\alpha}_F$ and $\bar{\alpha}_S$ are those which actually minimize $\bar{\bar{\eta}}_L$, $\bar{\theta}_F$ and $\bar{\theta}_S$, while the values of α_F^T and α_S^T are the values of α which minimized the corresponding execution times.

The entries in Table 4.2 show that α_η^e and α_S^e are quite effective as predictors of the α's which minimize $\bar{\bar{\eta}}_L$ and θ_S (and T_S). The formula for α_F^e was obtained by ignoring some terms in (4.8) which are relatively large for small n. As a consequence, α_F^e tends to be somewhat too large for small n. However, for moderately large n, α_F^e is a reliable predictor of the minimizing α. At any rate, errors of two or three in the predicted minimizing α are not very serious since the functions are quite flat near their minima. We present Table 4.3 as an example which demonstrates this phenomenon.

α	$\bar{\theta}_F$	T_F	$\bar{\theta}_S$	T_S	$\bar{\eta}_L$	$\bar{\bar{\eta}}_L$
1	1,394,939	23.63	131,120	1.70	65,560	67,160
2	2,389,535	29.84	134,548	1.66	33,779	35,619
3	2,069,316	27.71	95,816	1.21	25,438	27,518
4	1,749,871	24.15	78,924	1.13	22,557	24,877
5	1,572,862	22.72	70,768	.95	21,860	24,420
6	1,506,567	21.36	66,980	.87	22,255	25,055
7	1,437,720	20.93	65,688	.85	23,274	26,314
8	1,363,068	20.50	65,956	.86	24,683	27,963
9	1,377,468	21.16	67,472	.88	26,404	29,924
10	1,354,071	21.31	69,300	.91	28,203	31,963
11	1,399,535	22.36	72,064	.94	30,236	34,236
12	1,416,678	22.82	75,140	.97	32,347	36,587
13	1,433,821	23.48	78,216	1.04	34,458	38,938

Table 4.3 Actual operation counts and execution times for factorization and solution, the number of nonzero components of L stored, and the total storage associated with L, for the one-way dissection code with n = 40 and α = 1(1)13.

There are several apparent anomolies associated with $\bar{\theta}_F$ and T_F in Table 4.3. The first is that $\bar{\theta}_F$ does not appear to be quite convex; this is because some values of α lead to somewhat more "uniform" dissections than others, which result in a few irregularities which are apparent near the minimum. Second, $T_F(8) < T_F(10)$ even though $\bar{\theta}_F(8) > \bar{\theta}_F(10)$. This can be explained at least in part by the fact that bookkeeping overhead increases with α. Moreover, our timing routine is subject to the usual vagaries of modern operating systems, and can be in error by up to about 5 percent.

Finally, just as we did for the codes described in sections 2 and 3, we would like to examine the operations-per-second output of our code to determine whether our data structure is significantly penalizing the speed of the computation. Table 4.4 contains the ratios $T_F/\bar{\theta}_F$ and $T_S/\bar{\theta}_S$ for n = 10(5) 40 and the α which minimizes $\bar{\theta}_F$. The results are essentially the same for the α's which minimize η and $\bar{\theta}_S$. Comparing the entries in Table 4.4 with those in Tables 3.2 and 2.1, we conclude that our one-way dissection code is about as efficient as the other two codes.

n	n^2	$T_F/\bar{\theta}_F$	$T_S/\bar{\theta}_S$
10	100	2.1(-5)	2.0(-5)
15	225	1.9(-5)	1.6(-5)
20	400	1.7(-5)	1.5(-5)
25	625	1.7(-5)	1.4(-5)
30	900	1.7(-5)	1.4(-5)
35	1225	1.6(-5)	1.3(-5)
40	1600	1.5(-5)	1.3(-5)

Table 4.4 $T_F/\bar{\theta}_F$ and $T_S/\bar{\theta}_S$ for n = 10(5)40 and the α which minimizes $\bar{\theta}_F$.

§5 Comparison of the Codes

Now that we have reported on some of the individual characteristics of the orderings and the codes which utilize them, we now turn to the task of evaluating their relative merits. There are several issues which must be considered in any such comparison. First, if a given matrix problem is to be solved only once, then a comparison between two ordering strategies should include the cost of producing the ordering and initializing any data structures. On the other hand, if many problems having the same zero-nonzero structure are to be solved, it is reasonable to ignore the initialization costs in the comparison. Such a situation might occur, for example, in solving a nonlinear boundary value problem.

Another related issue is whether more than one right hand side is involved in the matrix problem. In the solution of some mildly nonlinear and time dependent problems, many systems having the same coefficient matrix must be solved. In these cases, the cost of solving the equations, given the triangular factorization, may be the dominant consideration, whereas for a single right hand side, the cost of the factorization typically dominates the cost of solving the triangular systems.

In our reporting, we have intentionally ignored the cost of actually producing the ordering and setting up the data structures. We do not suggest that these costs are in general unimportant. However, our ordering subroutines were coded especially for the problems under study, and we feel that their (relatively small) execution times have little general relevance. Indeed, reliable automatic schemes for producing "good" nested dissection orderings and one-way dissection orderings, even if they exist, are not generally available. Thus, all our conclusions are based on the assumption

that the ordering has been supplied. If the ordering costs vary a great deal, (which was not the case for our experiments), our conclusions may be altered. However, as noted above, situations occur where it may very well be reasonable to ignore the ordering and set-up costs in comparing the schemes.

Although our experiments have been performed for a special problem, we contend that if the orderings are provided, our results are indicative of the effectiveness of these dissection strategies applied to problems having less regular geometry. The linear equation solvers whose performance we are comparing do not exploit the shape of the domain (mesh) associated with the matrix problems. They have been used to solve irregular mesh problems, ordered in the same manner as described for our regular test problems.

Table 5.1 contains the total storage requirement η for the subroutines which use the three strategies under study. Tables 5.6 and 5.7 contain factorization times T_F and times required for solution T_S, given the factorization, for the same matrix problems.

n^2	natural ordering	nested dissection ordering	One-way dissection ordering		
			α chosen to minimize η	α chosen to minimize T_F	α chosen to minimize T_S
100	1292	1740	1010	1110	1110
225	4037	5218	2653	3173	3173
400	9182	9245	5222	5746	5746
625	17477	15400	8942	9329	9329
900	29672	25381	13796	14812	14812
1225	46517	35257	20093	22025	20837
1600	68762	46607	27638	31193	29540

Table 5.1 Total storage requirements η for the three strategies

n^2	natural ordering	nested dissection ordering	One-way dissection ordering		
			α chosen to minimize η	α chosen to minimize T_F	α chosen to minimize T_S
100	.15	.14	.22	.22	.22
225	.58	.54	.76	.76	.76
400	1.62	1.04	2.13	1.99	1.99
625	3.83	1.95	4.55	4.26	4.26
900	7.31	3.65	8.62	8.05	8.05
1225	13.15	5.27	14.39	13.23	13.37
1600	22.77	7.46	22.72	20.50	20.93

Table 5.2 Factorization times T_F for the three strategies

n^2	natural ordering	nested dissection ordering	One-way dissection ordering		
			α chosen to minimize η	α chosen to minimize T_F	α chosen to minimize T_S
100	.03	.05	.04	.04	.04
225	.10	.17	.09	.09	.09
400	.21	.23	.18	.17	.17
625	.43	.36	.31	.28	.28
900	.68	.71	.48	.42	.42
1225	1.08	.91	.67	.65	.64
1600	1.60	1.09	.92	.86	.85

Table 5.3 Solution times T_S for the three strategies

On the basis of some simple least squares approximations to the data provided in the tables, along with the estimates (3.4) and (4.7), our conclusions can be summarized in the following table.

Criterion which is most important	Natural ordering	One-way dissection	Nested dissection
Storage (S)	$0 \leq n^2 < 15$	$15 \leq n^2 \lesssim 70{,}000$	$n^2 \gtrsim 70{,}000$
Factorization time (T_F)	$0 \leq n^2 \lesssim 100$	-	$n^2 \gtrsim 100$
Solution time, given the factorization (T_S)	$0 \lesssim n^2 \lesssim 225$	$225 \lesssim n^2 \lesssim 4{,}000$	$n^2 \gtrsim 4{,}000$

Table 5.4 Choice of ordering strategy based on n^2 and the criterion which is most important.

§6 Concluding Remarks

There is an important qualification which must be made concerning our experiments in order that they interpreted fairly. As mentioned in section 5, the cost of actually producing the orderings and initializing the data structures for the different solvers has been ignored. We did this for two reasons: a) for our special test problem, these costs were not very significant and b) for more general meshes, the problem of producing "good" dissection orderings is not well understood, and reliable algorithms have not been developed. On the other hand, good algorithms do exist for producing banded orderings for mesh problems [4,11]. Moreover, it seems probable that even if good algorithms are developed for dissection orderings, they will be at least as costly as those for producing banded orderings. Thus, it is important to emphasize that the conclusions in Table 5.4 are based solely on the performance of the solvers.

This brings us to a second important point. The linear equation solvers were not specialized in any way to the test problem. Thus, in view of the substantial savings which are possible using these dissection orderings, we would contend that the development of efficient, reliable algorithms for producing dissection orderings is a desirable goal.

References

[1] Garrett Birkhoff and Alan George, "Elimination by nested dissection" in Complexity of Sequential and Parallel Algorithms (J.F. Traub, editor), Academic Press, New York, 1973, pp.221-269.

[2] James R. Bunch, "Analysis of the diagonal pivoting method", SIAM J. Numer. Anal. 8 (1971), pp.656-680.

[3] James R. Bunch, and D.J. Rose, "Partitioning, tearing and modification of sparse linear systems", J. Math. Anal. and Appl., to appear.

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

[5] George E. Forsythe and Cleve B. Moler, Computer Solution of Linear Algebraic Systems, Prentice Hall, Inc., Englewood Cliffs, N.J., 1967.

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

[7] Alan George, "An efficient band-oriented scheme for solving n by n grid", Proc. 1972 FJCC, AFIPS Press, 210 Summit Ave., Montvale, N.J., pp.1317-1321.

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

[9] Alan George, "Computer implementation of the finite element method", Stanford University Technical Report STAN-CS-208, 1971.

[10] Alan George, "The calculation of arithmetic and storage estimates for sparse matrix calculations using a symbolic algebra system", unpublished manuscript.

[11] 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., to appear.

[12] F.G. Gustavson, "Some basic techniques for solving sparse systems of equations", Sparse Matrices and their Applications, D.J. Rose and R.A. Willoughby eds., Plenum Press, New York, 1972.

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

[14] R.S. Martin and J.H. Wilkinson, "Solution of symmetric and unsymmetric band equations and calculation of eigenvectors of band matrices", Numer. Math., 9 (1967), pp.279-301.

[15] R.P. Tewarsen, Sparse Matrices, Academic Press, New York, 1973.

[16] James H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.

[17] J.H. Wilkinson and C. Reinsch, Handbook for Automatic Computation, Vol.II, Springer Verlag, N.Y., 1971.