Wide Quotient Trees for Finite Element Problems

Earl Zmijewski[1]
John R. Gilbert[2]

TR 85-673
April 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

1 AT&T Bell Laboratories Scholar. The work of this author was also supported in part by the National science Foundation under Grant DCR-81-05763.

2 The work of this author was supported in part by the National Science Foundation under Grant MCS-82-02948.

Wide Quotient Trees for Finite Element Problems

Earl Zmijewski[1]

John R. Gilbert[2]

Computer Science Department
Cornell University
Ithaca, New York 14853

April 16, 1985

Abstract

In solving the system of linear equations $Ax = b$ where A is an $n \times n$ large sparse symmetric postive definite matrix, one important objective is to minimize fill. One approach is to partition the matrix so that its corresponding quotient graph is a tree and then use block factorization techniques to solve the system. We examine several methods for generating valid quotient tree partitionings of grid graphs and find that those producing short wide quotient trees are superior for large enough graphs. We then give an algorithm for generating wide quotient tree partitionings of a more general class of graphs. Bounds on its storage and computational requirements are provided and compared to those of a generalized nested dissection algorithm.

[1]AT&T Bell Laboratories Scholar. The work of this author was also supported in part by the National Science Foundation under Grant DCR–81–05763

[2]The work of this author was supported in part by the National Science Foundation under Grant MCS–82–02948.

1 Introduction

Consider the system of linear equations

$$Ax = b$$

where A is an $n \times n$ large sparse symmetric positive definite matrix. We can solve for x by computing the Cholesky factor, L, of A (i.e. the lower triangular matrix L such that $A = LL^T$) and then solving the systems $Ly = b$ and $L^Tx = y$. Considerable computer time and storage can be saved by taking advantage of A's sparsity. However, L will not, in general, be as sparse as A. The set of positions that are nonzero in L and zero in A is known as *fill*. To reduce the amount of fill, one generally solves the equivalent system

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

for some $n \times n$ permutation matrix P. Since A is positive definite, no pivoting is required to maintain numerical stability and hence, we are free to choose P solely on the basis of fill and algorithm design considerations.

Numerous algorithms have been suggested to solve this problem. Several use complicated data structures to save only the nonzeros of L (e.g. nested dissection [G73,GL81]), while others employ simpler schemes that save some zeros but reduce the overall storage and computational overhead (e.g. reverse Cuthill-McKee [GL81]). What these methods have in common is that they all compute and save every nonzero of L. In this paper, we investigate a very different approach, namely, the method of refined quotient trees, which is originally due to George and Liu [GL78,GL81]. This method partitions A into blocks and factors one block at a time. Only the factors of the diagonal blocks are saved; those of the off-diagonal blocks are generated as needed. Hence, we need only concern ourselves with fill occurring in the diagonal blocks.

Our paper is organized as follows. Section 2 reviews block factorization techniques and introduces the method of refined quotient trees. In section 3, we present ways to generate valid quotient tree partitionings of grid graphs and estimate the storage and computational time they require. The results of some numerical experiments using these techniques and George's nested dissection algorithm [G73] are found in section 4. In section 5, we present an algorithm for finding quotient tree partitionings of graphs that have bounded degree and satisfy a $n^{1/2}$-separator theorem. We then compare its behavior to that of a generalized nested dissection algorithm. Our concluding remarks are found in section 6.

2 Background

2.1 Block factorization

We begin by reviewing techniques for factoring partitioned matrices [GL81]. Suppose A is partitioned as

$$A = \begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix},$$

where A_{11} and A_{22} are both square. The Cholesky factor of A is given by

$$L = \begin{pmatrix} L_{11} & 0 \\ W & L_{22} \end{pmatrix},$$

where $L_{11}^T L_{11} = A_{11}$, $W = A_{21} L_{11}^{-T}$, and

$$L_{22} L_{22}^T = \hat{A}_{22} = A_{22} - WW^T.$$

Note that WW^T can be written as either

$$(A_{21} L_{11}^{-T})(A_{21} L_{11}^{-T})^T \text{ or } A_{21}(L_{11}^{-T}(L_{11}^{-1} A_{21}^T)).$$

This suggests two distinct methods for computing $\hat{A}_{22}$. The first is to explicitly form $W = A_{21} L_{11}^{-T}$ and then WW^T, subtracting the result from A_{22}. The second method finds $\hat{A}_{22}$ a column at a time. Note that the i^{th} column of WW^T is $A_{21}(L_{11}^{-T}(L_{11}^{-1} a_i))$ where a_i is the i^{th} column of A_{21}. Thus, we can compute the columns of WW^T one at a time, subtracting each column from A_{22} as it is formed. That is, $\hat{A}_{22}$ can be found without explicitly forming W.

This provides us with two techniques for solving $Ax = b$. One way is to calculate the entire Cholesky factor of A, namely, L_{11}, L_{22}, and W, and use forward and backward elimination to find x. Alternatively, we can compute just L_{11} and L_{22}. Then, during the elimination process, products of the form $W^T z$ and Wz can be computed as $A_{21}(L_{11}^{-T} z)$ and $L_{11}^{-1}(A_{21}^T z)$, respectively. Since W can be considerably less sparse than A_{21}, the second approach to solving $Ax = b$ can require less storage and arithmetic operations than the first.

In order to solve large systems of equations, we would like to partition A into as many blocks as possible. The second scheme can be generalized to handle matrices that are partitioned into more than four blocks; however, the off-diagonal blocks of L will not necessarily have W's simple form. Thus, the process of recomputing these blocks as needed could be prohibitively expensive. George and Liu have proposed a method for partitioning a matrix so that the off-diagonal blocks of its factor are easily computed. Before examining this method, we will introduce some graph theoretic notation.

2.2 Refined Quotient Trees

A *graph*, $G = (V, E)$, consists of a set V of *vertices* and a set E of *edges*. An edge (v, w) is an unordered pair of distinct vertices. Vertices v and w are *adjacent* if $(v, w) \in E$. The adjacency set of a vertex v is $adj(v) = \{w \in V \mid (v, w) \in E\}$. Furthermore, if S is a set of vertices, its *adjacency set* is $adj(S) = \{w \in V - S \mid (v, w) \in E$ for some $v \in S\}$. Given a partition $P = \{V_1, \cdots, V_k\}$ of V, the *quotient graph* of G with respect to P is (P, F) where $(V_i, V_j) \in F$ if and only if $adj(V_i) \cap V_j \neq \emptyset$.

A *path of length* k from v to w is a sequence of vertices $v = v_0, v_1, \ldots, v_k = w$ such that $(v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k-1$ and the vertices $v_1, \ldots, v_k$ are all distinct. A *cycle* is a path with $v_0 = v_k$. If there exists a path between every pair of vertices of a graph, the graph is *connected.* A *tree* is a connected graph with no cycles. A quotient graph that happens to be a tree is called a *quotient tree.* Assume $G = (V, E)$ is connected. The *distance* $d(v, w)$ between vertices v and w is the length of the shortest path from v to w. The *eccentricity* of vertex v is defined as

$$e(v) = max\{d(v, w) \mid w \in V\}.$$

Given an initial vertex x, we can define a partitioning $P = \{P_0, P_1, \cdots, P_{e(x)}\}$ of V as follows.

$$P_0 = \{x\}, P_1 = adj(x), \text{ and } P_i = adj(P_{i-1}) - P_{i-2} \text{ for } i = 2, \cdots, e(x).$$

Such a partitioning is called a *rooted level structure*. Notice that the corresponding quotient graph is a simple path.

Let $A = (a_{ij})$ be an $n \times n$ symmetric matrix. Its structure can be represented by a graph $G = (V, E)$ where $V = \{v_1, \cdots, v_n\}$ and $(v_i, v_j) \in E$ if and only if $a_{ij} \neq 0$. Suppose we partition V into k sets. This is equivalent to partitioning A into k^2 blocks, A_{ij}, for $1 \leq i, j \leq k$. Let L_{ij} be the corresponding blocks of the Cholesky factor, L, of A. If this partitioning of V induces a quotient graph that is a tree, the off-diagonal blocks of L are simply $L_{ij} = A_{ij} L_{jj}^{-T}$. This observation is exploited in Sparspak's *refined quotient tree algorithm* (RQT) for solving linear systems [GL81]. The algorithm has two distinct parts, which we will denote as pRQT and fRQT. In the first part (pRQT), the matrix A is reordered and partitioned so that its corresponding quotient graph is a tree. pRQT attempts to find a partitioning with numerous members. It starts by locating a vertex v of high eccentricity. The level structure rooted at vertex v is then generated. Finally, each element of this partitioning is subdivided as much as possible. The second phase (fRQT) factors A and solves $Ax = b$. Only the diagonal blocks of L are calculated and saved. The off-diagonal blocks are computed as needed during the process of forward and backward solving.

We should mention that, for arbitrary graphs, the problem of finding a quotient tree partitioning with as many members as possible has been shown by Edenbrandt [E84] to be NP-complete. Hence, one might consider quotient tree partitionings with other properties. For example, one approach would be to minimize the size of the largest partition member. Another alternative would be to minimize the sum of the squares of the sizes of the partition members. This is equivalent to minimizing the total space occupied by the diagonal blocks in the corresponding matrix. However, for arbitrary graphs, these problems are also NP-complete [E84].

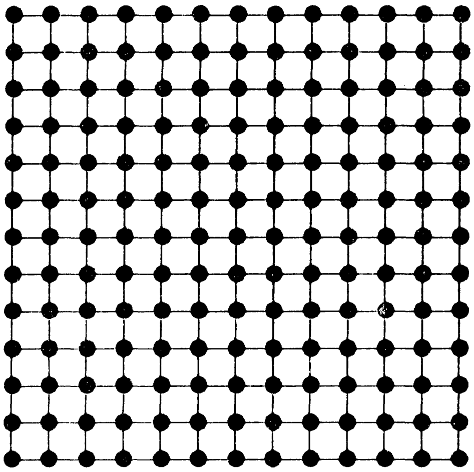

Figure 1. A 13×13 grid graph.

3 Partitionings for Grid Graphs

Let G be an $n \times n$ grid graph (see figure 1). We propose four methods of partitioning G to induce quotient trees. We then compare the various methods by estimating the operations and storage required to factor the corresponding matrices using fRQT. Examining the structure of G, several valid quotient tree partitionings are readily apparent. For example, we can partition G according to its *rows* (fig. 2) or its *diagonals* (fig. 3). The latter partitioning is the one found by pRQT. Besides these two techniques, we also consider two variants of nested dissection. The first one involves partitioning G into two parts using a separator consisting of the vertices on G's border plus those on a grid line that as closely as possible divides G into two equal squares. This process is then applied recursively to the two resulting components (fig. 4). In our second nested dissection method, each separator consists of the vertices on the border of a subgraph plus those on $k-1$ horizontal and $k-1$ vertical grid lines (for some fixed $k > 1$). The grid lines are selected to divide the graph into k^2 components that are roughly the same size and square (fig. 5). Notice that the quotient trees produced by methods one and two are long and narrow; they have height

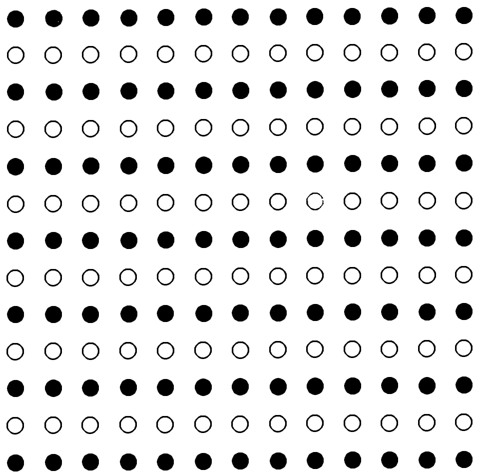

Figure 2. Method 1 applied to a 13×13 grid graph (the edges have been deleted for clarity).

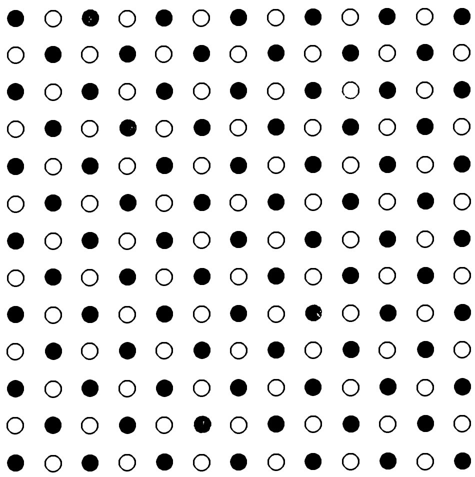

Figure 3. Method 2 applied to a 13×13 grid graph.

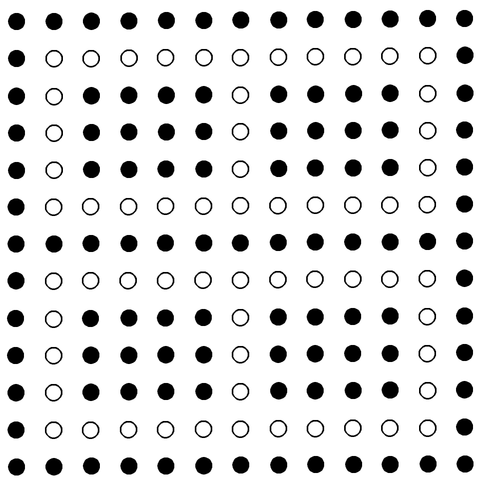

Figure 4. Method 3 applied to a 13×13 grid graph.

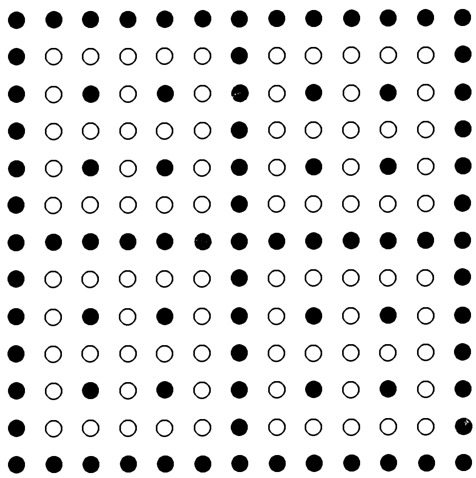

Figure 5. Method 4 with $k = 2$ applied to a 13×13 grid graph.

$O(n)$ and width 1. Those produced by methods three and four are short and wide with height $O(\log n)$ and width $O(n^2)$.

Before analyzing each of the proposed partitioning methods, we make several simplifying assumptions. First, we assume that the computation is dominated by the time required to factor the diagonal blocks. Further, we assume that these blocks are full and require essentially all of the storage needed to perform the factorization. These assumptions are not without some justification. For every off-diagonal block, fRQT modifies some diagonal block. Hence, the diagonal blocks can suffer a great deal of fill. In addition, the off-diagonal blocks suffer no fill since their factorizations are never explicitly computed.

We now estimate the operations and storage required by each of the methods suggested. Under our assumptions, an $i \times i$ diagonal block requires approximately $i^2/2$ storage locations and can be factored in $F(i) = i^3/6 + i^2/2 - 2i/3$ operations. In the first method (fig. 2), we have n diagonal blocks of size n. Hence, we require $n^3/2$ storage locations and $n \cdot F(n) = n^4/6 + n^3/2 - 2n^2/3$ operations to factor the corresponding matrix. The second technique (fig. 3) produces $2n - 1$ diagonal blocks and calls for

$$\tfrac{1}{2}n^2 + 2\sum_{i=1}^{n-1} \tfrac{1}{2}i^2 = \tfrac{1}{3}n^3 + \tfrac{1}{6}n \quad \text{storage locations and}$$

$$F(n) + 2\sum_{i=1}^{n-1} F(i) = \tfrac{1}{12}n^4 + \tfrac{1}{3}n^3 - \tfrac{7}{12}n^2 + \tfrac{1}{6}n \quad \text{operations.}$$

Examining the third strategy (fig. 4), we see that the number of vertices in the *top-level* separator of an $n \times m$ grid is

$$S(n,m) = 2 \cdot \max(n,m) + 3 \cdot \min(n,m) - 6.$$

Thus, the storage required to save the separators of G as dense blocks is given by $B(n,n)$, where

$$B(n,m) =$$
$$\begin{cases} \tfrac{1}{2}S(n,m)^2 + B(\lceil \tfrac{1}{2}(n-3) \rceil, m-2) + B(\lfloor \tfrac{1}{2}(n-3) \rfloor, m-2) & \text{if } n \le m; \\ \tfrac{1}{2}S(n,m)^2 + B(n-2, \lceil \tfrac{1}{2}(m-3) \rceil) + B(n-2, \lfloor \tfrac{1}{2}(m-3) \rfloor) & \text{otherwise.} \end{cases}$$

This can be approximated by

$$\tilde{B}(n,m) = \begin{cases} \frac{1}{2}S(n,m)^2 + 2\tilde{B}(\frac{1}{2}n, m) & \text{if } n \le m; \\ \frac{1}{2}S(n,m)^2 + 2\tilde{B}(n, \frac{1}{2}m) & \text{otherwise.} \end{cases}$$

Letting

$$H(n) = \tfrac{1}{2}S(n,n)^2 = \tfrac{1}{2}(5n-6)^2 = \tfrac{25}{2}n^2 - 30n + 18 \quad \text{and}$$
$$G(n) = \tfrac{1}{2}S(\tfrac{1}{2}n,n)^2 = \tfrac{1}{2}(\tfrac{7}{2}n-6)^2 = \tfrac{49}{8}n^2 - 21n + 18$$

we have

$$\begin{aligned} \tilde{B}(n,n) &= H(n) + 2\tilde{B}(\tfrac{1}{2}n, n) \\ &= H(n) + 2[G(n) + 2\tilde{B}(\tfrac{1}{2}n, \tfrac{1}{2}n)] \\ &= H(n) + 2G(n) + 4H(\tfrac{1}{2}n) + 8G(\tfrac{1}{2}n) + 16\tilde{B}(\tfrac{1}{4}n, \tfrac{1}{4}n) \\ &= \sum_{i=0}^{\log_2 n - 1} 2^{2i}[H(n/2^i) + 2G(n/2^i)] \\ \tilde{B}(n,n) &= 24\tfrac{3}{4}n^2 \log_2 n - 54n^2 + 72n - 18. \end{aligned}$$

Similarly, the number of operations needed to perform the factorization is given by $C(n,n)$, where

$$C(n,m) =$$
$$\begin{cases} F(S(n,m)) + C(\left\lceil \frac{1}{2}(n-3) \right\rceil, m-2) + C(\left\lfloor \frac{1}{2}(n-3) \right\rfloor, m-2) & \text{if } n \le m; \\ F(S(n,m)) + C(n-2, \left\lceil \frac{1}{2}(m-3) \right\rceil) + C(n-2, \left\lfloor \frac{1}{2}(m-3) \right\rfloor) & \text{otherwise.} \end{cases}$$

As with B, we can approximate C by

$$\tilde{C}(n,m) = \begin{cases} F(S(n,m)) + 2\tilde{C}(\frac{1}{2}n, m) & \text{if } n \le m; \\ F(S(n,m)) + 2\tilde{C}(n, \frac{1}{2}m) & \text{otherwise.} \end{cases}$$

Letting

$$H(n) = F(S(n,n)) = \tfrac{125}{6}n^3 - \tfrac{125}{2}n^2 + \tfrac{170}{3}n - 14 \quad \text{and}$$
$$G(n) = F(S(\tfrac{1}{2}n,n)) = \tfrac{343}{48}n^3 - \tfrac{245}{8}n^2 + \tfrac{119}{3}n - 14$$

we conclude that

$$\begin{aligned}\tilde{C}(n,n) &= \sum_{i=0}^{\log_2 n-1} 2^{2i}[H(n/2^i) + 2G(n/2^i)] \\ &= 70\tfrac{1}{4}n^3 - 123\tfrac{3}{4}n^2 \log_2 n + 51\tfrac{3}{4}n^2 - 136n + 14.\end{aligned}$$

In our final method (fig. 5), each separator consists of the border of some subgraph plus $2(k-1)$ grid lines that divide the graph into k^2 roughly equal parts (where $k > 1$ is fixed). Thus, the number of vertices in the top-level separator of G is

$$S_k(n) = 2(k+1)n - (k+1)^2.$$

The total storage required is given by

$$\begin{aligned}B_k(n) &= \sum_{i=0}^{\log_k n-1} [k^{2i} \cdot \tfrac{1}{2}S_k^2(n/k^i)] \\ &= 2(k+1)^2 n^2 \log_k n - [(k+1)^3/(k-1)](\tfrac{3}{2}n^2 - 2n + \tfrac{1}{2}).\end{aligned}$$

For $k = 2, 3,$ and 4 we have

$$\begin{aligned}B_2(n) &= 18n^2 \log_2 n - 40\tfrac{1}{2}n^2 + 54n - 13\tfrac{1}{2} \\ B_3(n) &= 32n^2 \log_3 n - 48n^2 + 64n - 16 \\ &\approx 20\tfrac{1}{5}n^2 \log_2 n - 48n^2 + 64n - 16 \\ B_4(n) &= 50n^2 \log_4 n - 62\tfrac{1}{2}n^2 + 83\tfrac{1}{3}n - 20\tfrac{5}{6} \\ &= 25n^2 \log_2 n - 62\tfrac{1}{2}n^2 + 83\tfrac{1}{3}n - 20\tfrac{5}{6}.\end{aligned}$$

Since $(k+1)^2/\log_2 k$ is increasing, the leading coefficient of $B_k(n)$ will increase with k.

Computing the factorization requires $C_k(n)$ operations where

$$\begin{aligned}C_k(n) &= \sum_{i=0}^{\log_k n-1} k^{2i} F(S_k(n/k^i)) \\ &= \tfrac{4}{3}n^3[k(k+1)^3/(k-1)] - 2n^2 \log_k n[(k+1)^4 - (k+1)^2] \\ &\quad + \tfrac{1}{6}n^2[5(k+1)^5 - 9(k+1)^3 - 4(k+1) - 8k(k+1)^2]/(k-1) \\ &\quad - \tfrac{1}{3}n[3(k+1)^5 - 6(k+1)^3 + 4(k+1)]/(k-1) \\ &\quad + \tfrac{1}{6}[(k+1)^5 - 3(k+1)^3 - 4(k+1)]/(k-1).\end{aligned}$$

Table 1 Comparsion of Methods

method	storage	operations
1	$\frac{1}{2}n^3$	$\frac{1}{6}n^4$
2	$\frac{1}{3}n^3$	$\frac{1}{12}n^4$
3	$24\frac{3}{4}n^2 \log_2 n$	$70\frac{1}{4}n^3$
$4, k = 2$	$18n^2 \log_2 n$	$72n^3$
$4, k = 3$	$20\frac{1}{5}n^2 \log_2 n$	$128n^3$
$4, k = 4$	$25n^2 \log_2 n$	$222\frac{2}{9}n^3$

Using this we find that

$$\begin{aligned}
C_2(n) &= 72n^3 - 144n^2 \log_2 n + 88n^2 - 193n + 25 \\
C_3(n) &= 128n^3 - 480n^2 \log_3 n + 249\tfrac{1}{3}n^2 - 418\tfrac{2}{3}n + 68 \\
&\approx 128n^3 - 302\tfrac{4}{5}n^2 \log_2 n + 249\tfrac{1}{3}n^2 - 418\tfrac{2}{3}n + 68 \\
C_4(n) &= 222\tfrac{2}{9}n^3 - 1200n^2 \log_4 n + 582\tfrac{2}{9}n^2 - 960\tfrac{5}{9}n + 151\tfrac{2}{3} \\
&= 222\tfrac{2}{9}n^3 - 600n^2 \log_2 n + 582\tfrac{2}{9}n^2 - 960\tfrac{5}{9}n + 151\tfrac{2}{3}.
\end{aligned}$$

Table 1 summarizes our results, ignoring lower order terms. We see that, for large enough n, method 3 minimizes operations, whereas method 4 with $k = 2$ needs the least storage. Since method 4, $k = 2$ requires only slightly more operations than method 3, we hypothesize that it will minimize total computer cost (where the computer cost is some function of storage and operations). The superiority of methods 3 and 4 results from the fact that, for large enough n, both partition the matrix into many more blocks than either methods 1 or 2.

Before considering some numerical results, we will propose an improvement to our two nested dissection techniques, methods 3 and 4. Note that the top-level separator of G produced by method 3 contains $5n-6$ vertices. Thus, to factor the corresponding block requires $F(5n-6) \approx 20\frac{5}{6}n^3$ operations or about 30% of the total. In addition, this block uses $\frac{1}{2}(5n-6)^2 \approx$

$12\frac{1}{2}n^2$ storage locations. If we could reduce the size of this initial separator while still generating a valid quotient tree ordering, the execution time could be significantly reduced. We suggest using an initial separator of $2n-1$ vertices consisting of one horizontal and one vertical mesh line that as closely as possible divides the graph into four equal squares. Method 3 or 4 can then be applied to the resulting subgraphs. For method 3, our storage requirement becomes

$$\tfrac{1}{2}(2n-1)^2 + 4\tilde{B}(\tfrac{1}{2}n, \tfrac{1}{2}n) = 24\tfrac{3}{4}n^2\log_2 n - 130\tfrac{3}{4}n^2 + 142n - 71\tfrac{1}{2},$$

while the number of operations needed is reduced to

$$F(2n-1) + 4\tilde{C}(\tfrac{1}{2}n, \tfrac{1}{2}n) = 36\tfrac{11}{24}n^3 - 123\tfrac{3}{4}n^2\log_2 n + 175\tfrac{1}{2}n^2 - 274\tfrac{1}{3}n + 57.$$

For method 4, $k=2$ we require

$$\tfrac{1}{2}(2n-1)^2 + 4B_2(\tfrac{1}{2}n) = 18n^2\log_2 n - 56\tfrac{1}{2}n^2 + 106n - 53\tfrac{1}{2}$$

storage locations and

$$F(2n-1) + 4C_2(\tfrac{1}{2}n) = 37\tfrac{1}{3}n^3 - 144n^2\log_2 n + 232n^2 - 388\tfrac{1}{3}n + 101$$

operations.

Unfortunately, it appears that the method of quotient trees is not the best one for this problem. In [G73], George showed that a matrix corresponding to an $n \times n$ grid graph can be factored using approximately $7\frac{3}{4}n^2\log_2 n$ storage locations and $9\frac{15}{28}n^3$ operations by employing a nested dissection ordering. This is considerably better than any of our proposed quotient tree methods.

4 Numerical Experiments

In order to verify the claims of the previous section, we used Sparspak's RQT routines to solve several systems of the form $Ax = b$, where A is the $n^2 \times n^2$ matrix corresponding to an $n \times n$ grid graph. A was ordered using method 2 or the improved version of either methods 3 or 4, $k=2$ and was factored by fRQT. Recall that method 2 is used by pRQT. To implement methods 3 and 4, we simply modified this routine. For each system that was successfully solved, the following statistics are reported: the total number of blocks generated by the ordering, the total number of storage locations required including overhead, the number of multiplicative operations needed to perform the factorization and carry out the process of forward and backward solving, and the time in seconds for both of these computions. We also include the factors by which the predicted requirements differ from the actual requirements. These factors appear in parentheses immediately after their corresponding values. The results are found in Tables 2, 3, and 4. We then solved the same systems using Sparspak's nested dissection routines. Table 5 contains these results. All experiments were run on a Vax 780 under Berkeley Unix at Cornell University.

Examining the first three tables, we see that both methods 3 and 4 require less storage than method 2 for all $n \geq 150$. In fact, for $n = 350$, method 4 needs less than $\frac{3}{5}$ the storage used by method 2. As predicted, method 4 requires more operations to compute the factorization than method 3, for all n. However, neither method beats the factorization time of method 2 until $n \geq 300$. This is due to the rather large initial blocks (i.e. separators) produced by methods 3 and 4. In general, the three quotient tree methods behaved as predicted. Turning our attention to the nested dissection results of Table 5, we see that, as expected, nested dissection required substantially less storage and operations than any of the proposed quotient tree methods, for all n.

5 A Partitioning Algorithm

We begin this section with a definition. Following Lipton and Tarjan [LT79], we say that a class of graphs S satisfies a *f(n)-separator theorem* for constants $\alpha < 1$ and $\beta > 0$ if the vertices of any n-vertex graph in S can be partitioned into three sets, A, B, and C, such that

$$\begin{aligned} |A|, |B| &\leq \alpha n, \\ |C| &\leq \beta f(n), \text{ and} \\ adj(A) \cap B &= \emptyset. \end{aligned}$$

Table 2 Method 2 applied to $n \times n$ grid graphs
(Storage and operations are scaled by 10^{-6})

n	number of blocks	total storage	operations fact	operations solve	time fact	time solve
100	199	0.448(1.345)	13.2(1.519)	1.4	280.7	26.5
150	299	1.384(1.230)	58.2(1.343)	4.6	1173.3	100.7
200	399	3.127(1.173)	170.8(1.256)	10.9	3197.8	233.1
250	499	5.927(1.138)	398.1(1.204)	21.2	7196.0	454.4
300	599	10.035(1.115)	799.9(1.170)	36.5	14281.1	784.9
350	699	15.700(1.099)	*	*	*	*

*(exceeded maximum storage available)

Table 3 Improved method 3 applied to $n \times n$ grid graphs.
(Storage and operations are scaled by 10^{-6})

n	number of blocks	total storage	operations fact	operations solve	time fact	time solve
100	253	0.476(1.356)	29.4(0.980)	1.5	560.7	31.7
150	509	1.294(1.171)	96.3(0.901)	4.2	1831.6	90.7
200	1021	2.609(1.103)	223.9(0.859)	8.7	4083.9	191.8
250	1021	4.426(1.058)	409.4(0.789)	14.9	7199.5	334.9
300	2045	6.896(1.044)	694.9(0.765)	23.5	12999.4	549.2
350	4017	9.971(1.033)	*	34.2	20142.2	800.7

*(value too large — not reported by Sparspak)

Table 4 Improved method 4, $k = 2$ applied to $n \times n$ grid graphs. (Storage and operations are scaled by 10^{-6})

n	number of blocks	total storage	operations fact	operations solve	time fact	time solve
100	341	0.465(0.725)	33.9(1.129)	1.4	606.0	25.9
150	1365	1.253(0.750)	116.0(1.076)	4.0	2024.5	85.9
200	1365	2.447(0.750)	259.9(0.985)	7.9	4469.2	174.1
250	5209	4.177(0.765)	484.8(0.923)	13.6	8435.8	312.0
300	5461	6.398(0.773)	813.6(0.882)	21.0	13942.7	476.0
350	5461	9.110(0.775)	*	30.2	21757.6	712.8

*(value too large — not reported by Sparspak)

Table 5 Nested dissection applied to $n \times n$ grid graphs. (Storage and operations are scaled by 10^{-6})

n	total storage	operations fact	operations solve	time fact	time solve
100	0.331	5.3	0.4	160.6	10.5
150	0.806	18.1	1.0	523.0	29.7
200	1.494	43.1	1.9	1310.6	73.9
250	2.412	84.3	3.1	2083.7	101.9
300	3.600	146.8	4.7	3466.8	161.3
350	5.003	233.7	6.6	5318.0	218.9

For what follows, assume that G and all of its subgraphs satisfy a $n^{1/2}$-separator theorem for some $\alpha < 1$ and $\beta > 0$. Furthermore, assume that the maximum degree of any vertex in G is d where $d \ll n$. Two classes of graphs for which these assumptions hold are planar graphs of bounded degree [LT79] and two-dimensional finite element graphs. We can find a wide quotient tree partitioning P of G using the following algorithm.

Partition($\tilde{G}$,$\tilde{C}$)

1. Find a separator C of $\tilde{G}$ containing at most $\beta n^{1/2}$ vertices that divides $\tilde{G}$ into components $G_1, \cdots, G_k$ of at most αn vertices each.
2. Add to C any vertices of $\tilde{G}$ adjacent to $\tilde{C}$.
3. Add C to the partition P.
4. Remove any vertices of C from G_i and call Partition(G_i,C) for $i = 1, \cdots, k$. (We call C the parent of the G_i's.)

Initially, we set $P = \emptyset$ and call Partition(G,$\emptyset$). It should be clear that Partition generates a valid quotient tree partitioning of G.

Assume the vertices of G have been partitioned by this algorithm. We estimate the operations and storage required by fRQT to factor the corresponding matrix under the same assumptions employed in section 3 (i.e. we consider only the factorization and storage of the diagonal blocks, which we assume are dense).

Suppose Partition is called with arguments $\tilde{G}$ and $\tilde{C}$, where $\tilde{G}$ is a subgraph of G and $\tilde{C}$ is $\tilde{G}$'s parent. Partition generates a new partition member, $\tilde{P}$, consisting of the vertices of a separator of $\tilde{G}$ plus those vertices of $\tilde{G}$ adjacent to $\tilde{C}$. Since at most $d|\tilde{C}|$ vertices $\tilde{G}$ are adjacent to $\tilde{C}$, $|\tilde{P}| \leq d|\tilde{C}| + \beta|\tilde{G}|^{1/2}$. Partition is then applied to the connected components of $\tilde{G}$ that result from removing the vertices of $\tilde{P}$. Thus, the maximum storage required to save the diagonal blocks induced by this partitioning is $B(n, 0)$, where

$$B(n, m) = \tfrac{1}{2}(\min(\beta n^{1/2} + m, n))^2 + \max\{B(n_1, \gamma_1) + \cdots + B(n_k, \gamma_k)\}$$

and the maximum is taken over all $k, n_1, \ldots, n_k, \gamma_1, \ldots, \gamma_k$ satisfying

$$n_1 + \cdots + n_k \le n, \quad 0 \le n_i \le \alpha n \quad \text{for } i = 1, \ldots, k \qquad \text{and}$$
$$\gamma_1 + \cdots + \gamma_k \le d\beta n^{1/2}, \quad 0 \le \gamma_i \le d\beta n^{1/2} \quad \text{for } i = 1, \ldots, k.$$

The first argument to B, namely n, represents the size of a subgraph of G, whereas the second argument, m, denotes the number of vertices in this subgraph adjacent to its parent.

Clearly, B can be bounded by

$$B(n, m) \le \tfrac{1}{2}(\beta n^{1/2} + m)^2 + \max\{B(n_1, \gamma_1) + \cdots + B(n_k, \gamma_k)\}.$$

Using this observation and the fact that

$$\tfrac{1}{2}(\beta n^{1/2} + m)^2 = \tfrac{1}{2}\beta^2 n + \beta m n^{1/2} + \tfrac{1}{2}m^2,$$

we can bound $B(n, 0)$. Let

$$\begin{aligned}\tilde{B}(n) = \tfrac{1}{2}\beta^2 n + \max\{\tilde{B}(n_1) + \cdots + \tilde{B}(n_k) + \beta\gamma_1 n_1^{1/2} + \cdots + \beta\gamma_k n_k^{1/2} \\ + \tfrac{1}{2}\gamma_1^2 + \cdots + \tfrac{1}{2}\gamma_k^2\}\end{aligned}$$

where the maximum is taken over the set used in defining B. It follows that $B(n, 0) \le \tilde{B}(n)$.

Now, suppose that the maximum in the definition of $\tilde{B}(n)$ is attained by $n_1, \ldots, n_k, \gamma_1, \ldots, \gamma_k$. Let

$$F(n_1, \ldots, n_k, \gamma_1, \ldots, \gamma_k) = \beta\gamma_1 n_1^{1/2} + \cdots + \beta\gamma_k n_k^{1/2} + \tfrac{1}{2}\gamma_1^2 + \cdots + \tfrac{1}{2}\gamma_k^2.$$

Without loss of generality, assume n_1 is the largest n_i and consider the following.

$$\begin{aligned}\beta(\gamma_1 + \gamma_i)n_1^{1/2} + \tfrac{1}{2}(\gamma_1 + \gamma_i)^2 &= \beta\gamma_1 n_1^{1/2} + \beta\gamma_i n_1^{1/2} + \tfrac{1}{2}\gamma_1^2 + \tfrac{1}{2}\gamma_i^2 + \gamma_1\gamma_i \\ &\ge \beta\gamma_1 n_1^{1/2} + \beta\gamma_i n_i^{1/2} + \tfrac{1}{2}\gamma_1^2 + \tfrac{1}{2}\gamma_i^2.\end{aligned}$$

Thus,

$$F(n_1, \ldots, n_k, \gamma_1 + \gamma_i, \gamma_2, \cdots, \gamma_{i-1}, 0, \gamma_{i+1}, \ldots, \gamma_k) \ge F(n_1, \ldots, n_k, \gamma_1, \ldots, \gamma_k)$$

for any $i \in \{2,3,\ldots,k\}$. We conclude that the maximum value of F can be obtained by letting $\gamma_1 = d\beta n^{1/2}$ and $\gamma_i = 0$ for $i = 2,\ldots,k$. Therefore, $\tilde{B}$ can be rewritten as

$$\tilde{B}(n) = \tfrac{1}{2}\beta^2 n + \max\{\tilde{B}(n_1) + \cdots + \tilde{B}(n_k) + \beta(d\beta n^{1/2})n_1^{1/2} + \tfrac{1}{2}(d\beta n^{1/2})^2\}$$

where the maximum is taken over all $k, n_1, \ldots, n_k$ satisfying

$$n_1 + \cdots + n_k \leq n,\ 0 \leq n_i \leq \alpha n, \text{ and } n_i \leq n_1 \text{ for } i = 1,\ldots,k.$$

Since $n_1 \leq \alpha n$, we have

$$\tilde{B}(n) \leq \tfrac{1}{2}\beta^2 n(1 + 2d\alpha^{1/2} + d^2) + \max\{\tilde{B}(n_1) + \cdots + \tilde{B}(n_k)\}.$$

As shown by Gilbert [G80, lemma 2.5.3], we can bound $\tilde{B}$ by

$$\tilde{B}(n) \leq \frac{\frac{1}{2}\beta^2(1 + 2d\alpha^{1/2} + d^2)}{-\alpha\log_2\alpha - (1-\alpha)\log_2(1-\alpha)} n\log_2 n.$$

Similarly, we can bound the number of operations needed to perform the factorization. Under our assumptions, we require $C(n,0)$ operations where

$$C(n,m) = \tfrac{1}{6}(\min(\beta n^{1/2} + m, n))^3 + \max\{C(n_1,\gamma_1) + \cdots + C(n_k,\gamma_k)\}$$

and the maximum is taken over the set used in defining B. As with B, we can bound $C(n,0)$ by $\tilde{C}(n)$ where

$$\begin{aligned}\tilde{C}(n) = \tfrac{1}{6}\beta^3 n^{3/2} + &\max\{\tilde{C}(n_1) + \cdots + \tilde{C}(n_k) \\ &+ \tfrac{1}{6}[2\beta^2\gamma_1 n_1 + \beta\gamma_1^2 n_1^{1/2} + \beta^2\gamma_1 n_1 + 2\beta\gamma_1^2 n^{1/2} + \gamma_1^3]\},\end{aligned}$$

$\gamma_1 = d\beta n^{1/2}$, and the maximum is taken over the set used in defining $\tilde{B}$. Since $n_1 \leq \alpha n_1$,

$$\tilde{C}(n) \leq \tfrac{1}{6}\beta^3 n^{3/2}(1 + 3\alpha d + 3\alpha^{1/2}d^2 + d^3) + \max\{\tilde{C}(n_1) + \cdots + \tilde{C}(n_k)\}.$$

Employing Gilbert's lemma again, we have that

$$\tilde{C}(n) \leq \frac{\frac{1}{6}\beta^3(1 + 3\alpha d + 3\alpha^{1/2}d^2 + d^3)}{1 - \alpha^{1/2}} n^{3/2}.$$

One important class of graphs with a $n^{1/2}$-separator theorem is that of planar graphs. Lipton and Tarjan [LT79] showed that planar graphs satisfy a $n^{1/2}$-separator theorem with constants $\alpha = \frac{2}{3}$ and $\beta = 8^{1/2}$. They also gave a linear-time algorithm for finding such separators. Hence, Partition could be implemented efficiently for this class of graphs. Djidjev [D82] lowered the constant β to $6^{1/2}$. For $\alpha = \frac{2}{3}$ and $\beta = 6^{1/2}$, we have

$$\tilde{B}(n) \leq 3.27(1 + 1.63d + d^2)n \log_2 n \qquad \text{and}$$
$$\tilde{C}(n) \leq 13.35(1 + 2d + 2.45d^2 + d^3)n^{3/2}.$$

Let QT denote a factorization algorithm that employs Partition and fRQT. We will compare QT to a factorization algorithm (GND) that uses the variation of Lipton, Rose, and Tarjan's generalized nested dissection algorithm found in [G80]. Theorem 2.6.1 and corollary 2.9.3 of [G80] tell us that GND can factor a matrix corresponding to G in $O(n^{3/2})$ operations with fill $F(n)$, where

$$F(n) \leq \frac{\frac{1}{2}\beta^2(1 + 2d\alpha^{1/2}/(1 - \alpha^{1/2}))}{-\alpha \log_2 \alpha - (1 - \alpha)\log_2(1 - \alpha)} n \log_2 n + O(dn).$$

Ignoring lower order terms, we see that the bound on $F(n)$ exceeds the bound on $\tilde{B}(n)$ whenever

$$\tfrac{1}{2}\beta^2(1 + 2d\alpha^{1/2} + d^2) < \tfrac{1}{2}\beta^2(1 + 2d\alpha^{1/2}/(1 - \alpha^{1/2})),$$

that is, whenever $d < 2\alpha/(1 - \alpha^{1/2})$. For example, taking $\alpha = \frac{2}{3}$ as in the planar separator theorem, this suggests that QT will require asymptotically less storage for $d \leq 7$. The important thing to note is that the bound on the storage required by QT is within a constant factor of the bound on the storage required by GND.

Observe that as $\alpha \to 1$, $2\alpha/(1 - \alpha^{1/2}) \to \infty$. Hence, as $\alpha \to 1$, the storage requirements of QT may be less than those of GND for graphs of larger and larger maximum degree. This makes sense if one considers the definition of α. Recall that G and all of its subgraphs satisfy a $n^{1/2}$-separator theorem for some $\alpha < 1$ and $\beta > 0$. Fix β and select the smallest α for which this statement is true. Now, if α is close to 1, a $\beta n^{1/2}$ separator of a subgraph of G may divide the vertices of the subgraph into very unequal parts. Thus, if we apply GND to G we may get a great deal of fill between separators. Since QT does not save this fill, its storage requirements may be less than those of GND. However, one can always decrease α by increasing β. This will result in an increase in the storage required by both GND and QT. For large graphs, such an increase may not be possible if we have limited virtual memory. In such cases, we may not be able to perform the factorization using GND.

6 Conclusions

We began by considering quotient tree partitionings of $n \times n$ grid graphs. Four partitioning techniques were proposed. Two of the methods, one of which is used by Sparspak, induce long narrow quotient trees and, under our assumptions, require $O(n^3)$ storage locations and $O(n^4)$ operations to perform the corresponding factorization. The remaining two methods produce short wide quotient trees and need $O(n^2 \log_2 n)$ storage locations and $O(n^3)$ operations. However, our numerical experiments showed that all four of these techniques are markedly inferior to nested dissection [G73] for large grids.

We then proposed an algorithm (Partition) for generating wide quotient tree partitionings of graphs of bounded degree that satisfy a $n^{1/2}$-separator theorem. The storage requirements of the associated factorization algorithm (QT) were shown to be within a constant factor of the storage requirements of the generalized nested dissection algorithm (GND) [G80]. The storage tradeoff seems to favor QT slightly for graphs of small degree; however, because GND will in general be much faster than QT, there seems to be little reason to favor quotient tree methods over generalized nested dissection.

We have ignored one important issue. If we are factoring a large matrix using a limited amount of physical memory in a paging environment, the paging behavior of the factorization algorithm becomes extremely important. In a companion report [ZG85] we show that, for a fixed physical memory size, if QT and GND are applied to the same matrix, they will produce roughly the same number of page faults.

References

[D82] Hristo Nicolov Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic and Discrete Methods* **3**: 229-240, 1982.

[E84] Anders Edenbrandt. Quotient tree partitioning of undirected graphs. Cornell University Computer Science Department technical report 84–654, 1984.

[G73] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis* **10**: 345-363, 1973.

[GL78a] Alan George and Joseph W. Liu. User guide for Sparspak: Waterloo Sparse Linear Equations Package. *Report CS–78–30*, Department of Computer Science, University of Waterloo, 1978.

[GL78b] Alan George and Joseph W. Liu. Algorithms for matrix partitioning and the numerical solution of finite element systems. *SIAM Journal on Numerical Analysis* **15**: 297-327, 1978.

[GL81] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite System.* Prentice-Hall, 1981.

[G80] John Russell Gilbert. *Graph Separator Theorems and Sparse Gaussian Elimination.* Ph. D. thesis, Stanford University, 1980.

[H69] Frank Harary. *Graph Theory.* Addison-Wesley, 1969.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics* **36**: 177-189, 1979.

[ZG85] Earl Zmijewski and John Russell Gilbert. An analysis of the paging behavior of quotient tree and nested dissection methods. To appear as a Cornell University Computer Science Department technical report, 1985.