# CUTTING DOWN ON FILL USING NESTED DISSECTION: PROVABLY GOOD ELIMINATION ORDERINGS*

AJIT AGRAWAL†, PHILIP KLEIN, AND R. RAVI‡

**Abstract.** In the last two decades, many heuristics have been developed for finding good elimination orderings for sparse Cholesky factorization. These heuristics aim to find elimination orderings with either low fill, low operation count, or low elimination height. Though many heuristics seem to perform well in practice, there has been a marked absence of much theoretical analysis to back these heuristics. Indeed, few heuristics are known to provide any guarantee on the quality of the elimination ordering produced for arbitrary matrices.

In this work, we present the first polynomial-time ordering algorithm that guarantees approximately optimal fill. Our algorithm is a variant of the well-known nested dissection algorithm. Our ordering performs particularly well when the number of elements in each row (and hence each column) of the coefficient matrix is small. Fortunately, many problems in practice, especially those arising from finite-element methods, have such a property due to the physical constraints of the problems being modeled.

Our ordering heuristic guarantees not only low fill, but also approximately optimal operation count, and approximately optimal elimination height. Elimination orderings with small height and low fill are of much interest when performing factorization on parallel machines. No previous ordering heuristic guaranteed even small elimination height.

We will describe our ordering algorithm and prove its performance bounds. We shall also present some experimental results comparing the quality of the orderings produced by our heuristic to those produced by two other well-known heuristics.

**1. Introduction.** Solution of linear systems of the form $Ax = b$ is a basic tool in numerical analysis and is commonly used in almost all branches of science and engineering. The most popular direct method of solving a system of linear equations is Gaussian elimination, in which the matrix $A$ is first transformed into an upper triangular matrix by forward elimination, and then the solution is found by backward substitution. In many applications, the coefficient matrix $A$ is sparse, i.e. has very few non-zero elements. Since the computations on the zero elements in $A$ can be performed trivially without requiring any floating point operations, it is important to take advantage of the sparsity to keep the computational complexity low. However, as the elimination phase progresses, new non-zero elements are introduced into the coefficient matrix, and the matrix tends to become less sparse. The loss in sparsity translates into an increased storage requirement, and also increased computation time since the non-zero elements enter into subsequent calculations.

The new non-zero elements introduced during the elimination process are called

* Some of the work reported in this paper first appeared in an extended abstract in the Proceedings of the 31st Annual IEEE Conference on the Foundations of Computer Science, 1990 [33].

† Digital Equipment Corp., Massively Parallel Systems Group, 146 Main Street, Maynard, MA 01574.

‡ Brown University, Providence, RI 02912. Research supported by NSF grant CCR-9012357 and an NSF PYI award, together with PYI matching funds from Thinking Machines Corporation and Xerox Corporation. Additional support provided by ONR and DARPA contract N00014-83-K-0146 and ARPA Order No. 6320, Amendment 1.

A. George et al. (eds.), *Graph Theory and Sparse Matrix Computation*

*fill-in.* Different orders of eliminating the variables may yield very different fill-in. It is thus of prime importance to be able to choose an elimination ordering of the variables that results in small fill-in. The choice of an elimination ordering that results in small fill-in often conflicts with the requirement of an ordering that ensures numerical stability of the solution process. Fortunately, many of the systems of equations that arise in practice are *positive definite*, in which numerical stability is not a problem [17]. In solving such linear equations, we are free to choose an ordering of variables entirely based on our desire to preserve the sparsity of the matrix during the elimination process.

A matrix $A$ is called *symmetric* if $A_{ij}$ equals $A_{ji}$ for every $i, j$. Positive definite matrices that are also symmetric frequently arise in structural analysis, signal processing, economics, VLSI simulation, solution of linear programs, and solution of partial differential equations, to name a few. In this work, we study the problem of finding a good elimination ordering for such matrices.

Henceforth, when we refer to a linear system of equations $Ax = b$, we assume that $A$ is a symmetric positive definite matrix.

**Minimizing fill.** We shall define the *fill* for an ordering as the sum of the number of non-zero elements in the matrix, and the fill-in introduced by the ordering. The fill for an ordering measures the amount of storage required, and also has bearing on the total time required for the elimination process. It is thus of interest to find an elimination ordering that minimizes fill. For the purposes of analyzing fill, we shall count each pair of symmetric elements only once.

Finding such an ordering is NP-complete [58] and hence is unlikely to have a polynomial-time solution. Nevertheless, this problem is of fundamental importance and a large set of ordering heuristics have been developed [3, 55, 14, 20, 49, 16, 15, 43, 18, 17, 9, 10]. However, none of them are known to give any performance guarantee on the size of the fill for arbitrary symmetric matrices. In this work, we present the first polynomial-time algorithm that guarantees approximately optimal fill. Our algorithm performs particularly well when the number of elements in each row (and hence each column) of the matrix is small. Fortunately, many problems in practice, especially those arising from finite-element methods, have such a property due to the physical constraints of the problems being modeled. As stated earlier, all our results are for the class of symmetric positive definite systems of equations.

THEOREM 1.1. *There is a polynomial-time algorithm that finds an elimination ordering yielding approximately optimal fill. The fill for the ordering is within a factor of* $O(\sqrt{d}\log^4 n)$ *of the optimum, where* $n$ *is the number of variables and* $d$ *is the maximum number of non-zero entries in any row or column of the coefficient matrix.*

Our algorithm is a variant of the well-known nested dissection algorithm [14]. It treats the input matrix as the adjacency matrix of a graph. The algorithm is based on finding a recursive decomposition of the graph associated with the coefficient matrix. The use of graphs in the study of elimination ordering is not new [51, 54]. There is an obvious way of associating a graph with a given symmetric matrix; the variables of the matrix associate with the nodes of the graph, and there is an edge between nodes

$i$ and $j$ iff the element $(i,j)$ of the matrix is non-zero. The values of the non-zero elements in the coefficient matrix are not relevant for the ordering problem in the case of symmetric positive definite systems.

Parter [51] first showed how to interpret the elimination process as a graph-theoretical process. Rose [55] characterized the class of graphs for which there is an elimination order with no fill-in, and showed how to find such an ordering. George [14] first proposed a nested dissection approach, designed specifically for grid graphs. The algorithm was shown to be optimal (within constant factors) in terms of operation count for the case of a regular finite-element mesh [29]. George's approach was first generalized to arbitrary graphs by George and Liu [16]. However they used a simple heuristic for graph separators and hence could not prove any bounds on its performance. Lipton, Rose, and Tarjan [41] proved a performance bound for their version of nested dissection algorithm for graphs with small ($O(\sqrt{n})$ in size) separators. Gilbert and Tarjan [26] later showed that by using small separators, George and Liu's nested dissection algorithm also gives close to optimal fill for planar graphs, but does not generalize to the class of graphs with $O(\sqrt{n})$ size separators. A good overview of these algorithms can be found in the books of George and Liu [17] and Duff and Reid [9].

Node separators are fundamental to the nested dissection algorithm. A node separator consists of a set of nodes whose removal breaks the graph up into pieces. For our purposes, every subset of the nodes is a separator. A separator is *f-balanced*, for some $f < 1$, if no piece on its removal has more than $fn$ nodes, where $n$ is the number of nodes in the graph.

The fill resulting from applying the approach of Lipton, Rose, and Tarjan depends upon the size of the separators for the class of graphs being dealt with. For example, their approach yields $O(n \log n)$ fill for any planar graph based on the fact that planar graphs have $\frac{2}{3}$-balanced separators of size $O(\sqrt{n})$.

There are three main differences between our work and the work of Lipton, Rose and Tarjan. One, we do not assume the existence of any special separator structure in the graphs. Two, our analysis is more strict; we are able to analyze the quality of our result with respect to the minimum fill achievable over all orderings. Three, our variation of nested dissection is similar to that of George and Liu, and somewhat simpler than that of Lipton, Rose and Tarjan.

Gilbert [22] showed that for a matrix with at most $d$ non-zero elements in each row (and column), there exists a nested dissection algorithm whose fill is within $O(d \log n)$ of minimum. His nested dissection algorithm, however, is inherently non-constructive; the choice of separators in his algorithm depends crucially on the optimally filled matrix.

Our nested dissection algorithm also uses balanced node separators. No polynomial time algorithms are known for finding a minimum-size balanced node separator in a graph. However, we show that choosing near-optimal balanced node separators is sufficient to achieve near-optimal fill. All our proofs are independent of the method by which the near-optimal separators are found. To our knowledge, Leighton and Rao [36, 37] have provided the first and the only polynomial-time algorithm to find an approximately balanced separator in an arbitrary graph. We hence use their method

in our nested dissection algorithm. However, in practice, other separator algorithms may be preferable on grounds of efficiency.

Though the performance guarantee of our algorithm deteriorates in going from a graph of small degree to one without this restriction, it is the first such algorithm that gives any non-trivial performance bound on the quality of the fill.

THEOREM 1.2. *There is a polynomial-time algorithm that produces an elimination ordering yielding a fill of* $O(F^* \frac{3}{4}\sqrt{m}\log^{3.5} n)$, *where* $F^*$ *is the size of the minimum fill, and* $m$ *is the number of non-zero elements in the given* $n \times n$ *matrix.*

Note that the performance guarantee for fill from the above theorem is never worse than $O(m^{\frac{1}{4}} \log^{3.5} n)$ since the size of the minimum fill $F^*$ must be at least as much as the number of edges $m$ in the graph.

**Minimizing the operation count.** We show that the ordering generated by our algorithm approximately minimizes not only fill, but also the total number of arithmetic operations required to solve the system of equations using Gaussian elimination. Much of the previous work in elimination ordering has been concerned with minimizing fill, and surprisingly little attention has been given to minimizing the number of operations. The only exceptions we know of are the works of Hoffman et al. [29], Lipton et al. [41] and Gilbert and Tarjan [26], who analyzed the operation counts for their nested dissection algorithms. However, their results only apply to specific classes of problems as explained in the discussions above.

THEOREM 1.3. *The elimination ordering produced by our algorithm approximately minimizes the total number of arithmetic operations. The performance guarantee is* $O(d \log^6 n)$ *for an* $n \times n$ *matrix with a maximum of* $d$ *non-zero elements in each row and column.*

**Solving sparse systems in parallel.** With the advent of parallel machines, much attention has naturally been focussed on solving sparse systems in parallel. In a typical parallel implementation of the elimination process, multiple variables are eliminated simultaneously in a step. Such variables must then have no dependencies between them, i.e. there must be no edge in the graph between the nodes corresponding to the variables. For an elimination ordering, the minimum number of parallel steps required to eliminate all the variables is called its *height* [47]. It is hence of interest to find an elimination ordering of minimum height. This problem is also known to be NP-complete [53]. Many researchers have given heuristics without any guarantees for this problem in the past [30, 38, 44, 48, 40, 10]. We prove that our nested dissection algorithm can guarantee approximately minimum height.

THEOREM 1.4. *The elimination ordering produced by our algorithm has height within a factor of* $O(\log^2 n)$ *of optimal.*

Note that our algorithm itself is not parallel, but is to be used to generate an ordering with small height. Such an approach is suitable for problems where the sparsity structure of the matrix is fixed, and the linear system has to be solved for many different coefficient values and/or right hand sides. A good elimination ordering can hence be found sequentially as a preprocessing step. Our work is thus different

TABLE 1

*Performance guarantees for the elimination ordering produced by our algorithm. In the above table, $n, m$ and $d$ respectively denote the number of nodes, edges, and the maximum degree of the graph associated with the coefficient matrix.*

| *Characteristic* | *Performance Guarantee* |
|---|---|
| Fill | $O\left(\min(\sqrt{d}\log^4 n, m^{\frac{1}{4}}\log^{3.5} n)\right)$ |
| Operation Count | $O(d\log^6 n)$ |
| Elimination Height | $O(\log^2 n)$ |

from other work addressing the issue of generating the ordering itself in parallel [43, 52, 39, 27, 25, 5].

Along with minimizing the height, it is desirable to keep both the fill and the operation count small, since they determine the total space and the total work done by the algorithm. Our algorithm is the first one known that approximately minimizes all three quantities *simultaneously*: fill, operation count, and height. By putting Theorems 1.1, 1.3 and 1.4 together, we get the following result.

THEOREM 1.5. *The elimination ordering produced by our algorithm simultaneously minimizes height to within a $O(\log^2 n)$ factor, fill to within $O(\sqrt{d}\log^4 n)$ factor, and the operation count to within a $O(d\log^6 n)$ factor of the respective optimum quantities. In the guarantee, $d$ denotes the maximum number of non-zero elements in any row or column of the $n \times n$ coefficient matrix.*

Bodlaender et al. [2] have independently presented essentially the same algorithm as ours for finding an elimination ordering of approximately minimum height. However, they do not analyze the fill and operation count for their ordering.

The performance guarantees for the elimination ordering obtained by our algorithm are given in Table 1.

Gilbert [22] has conjectured that there is an ordering that simultaneously minimizes height and approximately minimizes fill (to within a constant factor). Our analysis here represents progress towards proving this conjecture. Our algorithm is a polynomial-time algorithm since we utilize near-optimal node separators in our nested dissection algorithm. By utilizing a minimum node separator (non-polynomial) algorithm, we can prove the following:

THEOREM 1.6. *There exists a nested dissection ordering that simultaneously minimizes height to within a $O(\log n)$ factor, fill to within a $O(\sqrt{d}\log^2 n)$ factor, and operation count to within a $O(d\log^4 n)$ factor of the respective optimum quantities. In the guarantee, $d$ denotes the maximum number of non-zero elements in any row or column of the $n \times n$ coefficient matrix.*

**Chordal graphs.** An elimination step can be interpreted as updating the associated graph by adding new edges; the fill-in introduced by eliminating a variable $i$ turns the higher numbered neighbors of node $i$ into a clique. At the termination of the elimination process, the edges of the associated graph thus obtained correspond directly to the fill yielded by the ordering. Rose [55] showed that this updated graph is in fact *chordal*, and that minimizing fill corresponds to finding a minimum-size

chordal graph containing the graph associated with the input matrix. A *chordal* graph is a graph in which every cycle of length at least four has a chord, i.e. there is an edge between two non-consecutive nodes in the cycle. Chordal graphs are also sometimes referred to as *triangulated* graphs.

We exploit the characterization of the elimination process given by Rose. We prove that our nested dissection ordering yields a chordal graph of small size with respect to the optimal.

**Organization of the paper.** We explain the relationship between the elimination process and graph chordalization in Section 3. We give our algorithm in Section 4, and explain our algorithm in terms of a separator tree in Section 5. The lower and upper bounds for the fill, operation count, and height are provided in Sections 6, 7 and 8 respectively. We provide some experimental results for the performance of our algorithm in Section 9, and conclude the paper by discussing some open issues in Section 10.

**2. Notation.** A graph $G$ and a matrix $A$ are said to be *corresponding* if $(u, v) \in G$ if and only if $A_{uv} \neq 0$. Very often we shall refer to a matrix as a graph, meaning the graph corresponding to the matrix. The position of a node $v$ in an elimination ordering $\alpha$ will be denoted by $\alpha(v)$. Throughout this paper, we shall use $G^*_\alpha$ to refer to the chordal extension of a graph $G$ for a given ordering $\alpha$. Moreover, we shall refer to the optimal chordal extension of a graph $G$ by $G^*$; the optimality criteria for the extension will be clear from the context. By $|G|$ for a graph $G$, we shall refer to its number of edges.

**3. Graph chordalization.** An elimination ordering of a graph $G$ is said to be *perfect*, if the ordering induces zero fill-in on the matrix corresponding to the graph. A graph is *chordal* if it has a perfect elimination order. One of the simplest characterizations of a chordal graph is that every cycle of length at least four in the graph has an edge between some pair of non-consecutive nodes of the cycle. A good discussion of chordal graphs can be found in Golumbic's book [28]. Apart from Gaussian elimination, study of chordal graphs has applications in pedigree analysis, and evidence propagation in belief networks [31].

Graph chordalization is the problem of extending an input graph $G$ to a chordal graph by adding a minimum number of edges. Every minimal chordal extension of a graph can be completely specified by an ordering of the nodes of the graph; the ordering is a perfect elimination ordering for the chordal extension of the graph. For the matrix corresponding to the graph, this ordering of nodes also corresponds to an elimination ordering of the corresponding variables.

To construct a chordal extension of a graph $G$ from a given ordering $\alpha$ of its nodes, we mimic the elimination process in terms of the following operations on $G$ [54]. $G_0$ is set to be the original graph $G$. Let $G_i$ be the graph at the end of step $i$, and let $v$ be the $(i+1)$th node in the ordering. At step $i+1$, we augment $G_i$ to obtain $G_{i+1}$ by turning all the neighbors of $v$ numbered higher than $i+1$ into a clique. $G_n$ is the desired unique chordal graph corresponding to the ordering $\alpha$, where $n$ is the number of nodes in the graph.

An ordering of the nodes of a graph is hence sufficient to specify a chordal extension of a graph. We employ this approach in presenting our algorithm for approximately minimum chordalization.

## 4. Our algorithm

**Graph separators.** Before we give the ordering algorithm, we need some background on an essential ingredient of our algorithm, namely balanced node separators. Recall that a set of nodes $X$ in a graph $G = (V, E)$ is called an *$f$-balanced node separator* for some fraction $f < 1$, if no connected component of $G - X$ is of size more than the fraction $f$ of $|V|$. No polynomial-time algorithms are known for finding an $f$-balanced node separator of minimum size for a non-trivial constant $f$. However, using the technique of Leighton and Rao [36], one can find an approximately minimum-sized balanced node separator. This was also shown by Makedon and Tragoudas [50].

LEMMA 4.1 ([36, 50]). *There exists a polynomial-time algorithm to find a $\frac{2}{3}$-balanced node separator in a graph of size within an $O(\log n)$ factor of the optimal $\frac{1}{2}$-balanced node separator.*

Note that every $\frac{1}{2}$-balanced node separator is also a $\frac{2}{3}$-balanced node separator.

**Elimination ordering algorithm.** Our ordering algorithm is a nested dissection algorithm that is based on a recursive decomposition of the graph.

Given a graph $G = (V, E)$ with $n$ nodes, we proceed as follows to number its nodes in the range $[a, b]$, where $b = a + n - 1$. If $n = 1$, we number the single node $a$. Else, we find an approximate $\frac{2}{3}$-balanced node separator $X$ for $G$ using the algorithm in Lemma 4.1. We number the vertices in the separator from $b - |X| + 1$ to $b$ in any order. The rest of the nodes are numbered as follows. Let $G - X$ have $k$ connected subgraphs $A_1, \ldots, A_k$ of sizes $n_1, \ldots, n_k$. We recursively number the graph $A_i$ in the range $[a + \sum_{j=1}^{i-1} n_j, a - 1 + \sum_{j=1}^{i} n_j]$ for each $i \in [1, k]$.

We shall refer to this ordering as $\alpha$ for the rest of this paper.

**5. Separator tree.** In this section, we will establish a lower bound on the size of the optimum chordal extension of a graph in terms of the separator sizes found by the nested dissection algorithm. In our nested dissection ordering, we employ an approximation algorithm for finding balanced node separators. However, for ease of exposition below, we shall assume that we have a separator algorithm that finds the best balanced node separator in a graph. We shall later forego this assumption.

Consider the following tree, called the *separator tree*, representing the nested dissection process on a graph; the separator vertices form the root of the tree, and the trees of each of the pieces are built recursively. To distinguish from the nodes of the tree, we shall refer to the nodes of the graph as *vertices*. A tree node hence stands for a separator that may consist of several vertices. Our algorithm thus defines an elimination ordering of the vertices of the original graph that is consistent with a postorder traversal of the nodes of the separator tree. However, the algorithm orders the vertices within a tree node arbitrarily.

Let $G(V, E)$ be an input graph and $G^*$ be a minimum-size chordal extension of $G$. Let $T$ be the separator tree given by our nested dissection ordering (employing an optimal balanced separator algorithm). With each node $x$ in the separator tree $T$, let us associate three quantities $S_x$, $V_x$, and $G_x$. $S_x \subseteq V$ is the set of vertices forming the node $x$, $V_x \subseteq V$ is the set of vertices belonging to the nodes of the subtree rooted at $x$, and $G_x \subseteq G$ is the subgraph induced by the vertex set $V_x$ in the original graph $G$. Let us denote the separator containing a vertex $v$ by $X_v$ and the set of vertices belonging to any of the nodes in the subtree rooted at $X_v$ by $T_v$. The separator tree naturally defines an ancestor relation on the nodes of the tree. We shall also consider a node to be a trivial ancestor of itself. We shall say that a node $u$ is a *proper ancestor* of a node $v$ if $u$ is an ancestor of $v$ but $u \neq v$.

We shall derive a lower bound on the size of the optimal chordal extension $G^*$ in terms of the sizes of the separators at any level of the separator tree. By a level of the tree we mean all the nodes in the tree that are at the same distance from the root. Let $x_1, \ldots, x_p$ be the tree nodes at some level of the separator tree. Since $V_{x_1}, \ldots, V_{x_p}$ are disjoint, it follows that the graphs $G^*_{x_1}, \ldots, G^*_{x_p}$ induced by them in $G^*$ are also disjoint. Thus we have

$$\sum_{i=1}^{p} \left|G^*_{x_i}\right| \leq |G^*| \tag{1}$$

where $|G|$ refers to the number of edges (size) of $G$.

We shall now use the following two previously known results.

FACT 5.1. *Every node-induced subgraph of a chordal graph is chordal.*

Gilbert, Rose, and Edenbrandt [24] showed that every chordal graph has a balanced clique separator, i.e. a set of nodes that along with being a balanced node separator, induces a clique in the chordal graph. Since the number of edges in this clique can be at most the number of edges in the chordal graph, the following theorem follows.

THEOREM 5.1 ([24]). *Every chordal graph has a $\frac{1}{2}$-balanced clique separator, and hence has a $\frac{1}{2}$-balanced node separator of size at most $\sqrt{2E}$, where $E$ is the number of edges in the chordal graph.*

By Fact 5.1, each of the graphs $G^*_{x_i}$ is chordal. Hence by Theorem 5.1, we can write

$$|V_{x_i}|^2 \leq 2\left|G^*_{x_i}\right| \tag{2}$$

On rewriting (1) using this observation, we have the following lemma.

LEMMA 5.2. *The size of the optimal chordal extension is at least one-half the largest sum of the squares of the sizes of the separators at any level of the nested dissection separator tree.*

One of the main results of this work is to show that the nested dissection algorithm in fact yields a chordal graph whose size is close to the lower bound given above (see Section 6.3 for proof).

THEOREM 5.3. *For any level $l$ of the nested dissection separator tree, let $S_l$ be the sum of the squares of the sizes of the separators at this level. Then the size of the optimal chordal extension of $G$ is at least $\frac{1}{2}\max_l S_l$, and at most $O(\sqrt{d}\log^4 n)*\max_l S_l$.*

In employing an approximation algorithm for finding balanced node separators that has a factor of $f$ performance guarantee, we prove that the size of the chordal graph thus obtained is no more than $O(f^2)$ times the size of that obtained by using the optimal balanced node separators. We employ a separator algorithm with an $O(\log n)$-factor performance guarantee, and obtain the following result (see Section 6.3 for proof).

THEOREM 5.4. *There is a polynomial-time algorithm that generates a nearly optimal chordal extension of an input graph. The size of the chordal graph is $O(\min(|G^*|\sqrt{d}\log^4 n, |G^*|^{\frac{3}{4}}\sqrt{m}\log^{3.5} n))$, where $|G^*|$ is the size of the optimal chordal extension of an input graph of $n$ nodes, $m$ edges, and maximum degree $d$.*

**6. Performance guarantee: Number of edges.** In this section, we establish the performance guarantees for the number of edges and hence the fill for our elimination ordering.

**6.1. A lower bound.** We shall first establish a lower bound on the number of edges in the optimally filled graph $G^*$. In Lemma 5.2, we showed a lower bound for $|G^*|$ in terms of the sizes of the separators at any level of the separator tree. However, we had assumed that we had an optimal separator algorithm. We now relax that restriction, and derive a similar result using the nested dissection tree built with the $O(\log n)$ separator approximation algorithm of Leighton and Rao (see Lemma 4.1).

We first state the following simple observation.

PROPOSITION 6.1. *Let $x_1, \ldots, x_p$ be the separators at some level of the separator tree. The vertex sets $T_{x_1}, \ldots, T_{x_p}$ of the subtrees rooted at these separators are disjoint.*

LEMMA 6.2. *Let $X_1, \ldots, X_p$ be the separators at any level of the nested dissection separator tree. The size of the optimal chordal extension is $\Omega\left(\frac{\sum_{i=1}^{p}|X_i|^2}{\log^2 n}\right)$.*

*Proof.* Let $G_i^*$ be the subgraph induced in $G^*$ by the vertices belonging to the subtree rooted at $X_i$. By Theorem 5.1, $G_i^*$ has a $\frac{1}{2}$-balanced separator of size at most $\sqrt{2\,|G_i^*|}$. Let this separator be $X_i^*$. Then we have

$$\begin{aligned} \sum_{i=1}^{p}|X_i^*|^2 &\leq 2\sum_{i=1}^{p}|G_i^*| \\ &\leq 2|G^*| \end{aligned} \tag{3}$$

The second inequality follows from the disjointness of the subgraphs $G_i^*$, using Proposition 6.1. Let the graph induced in $G$ by the vertices of $G_i^*$ be $G_i$. Since the edges of $G_i$ form a subset of the edges of $G_i^*$, it follows that $X_i^*$ is also a $\frac{1}{2}$-balanced separator in $G_i$. By construction, the vertex set $X_i$ is a $\frac{2}{3}$-balanced node separator in $G_i$, and on applying Theorem 4.1, we have

$$|X_i| = O(\log n)|X_i^*|$$

which implies that

$$\sum_{i=1}^{p} |X_i|^2 \leq O(\log^2 n) \sum_{i=1}^{p} |X_i^*|^2 \tag{4}$$

On substituting (3) into (4), we get

$$\sum_{i=1}^{p} |X_i|^2 \leq O(\log^2 n)|G^*| \tag{5}$$

Hence the lemma follows on rewriting the last equation. □

**6.2. A characterization of chordal graphs.** Our aim is to estimate the number of edges in the chordal graph corresponding to the ordering given by our algorithm. To do so, we need a good characterization of these edges. Earlier we discussed one such characterization by specifying how to extend a graph to be chordal given the elimination ordering of its nodes. However, there is in fact a more direct characterization of these edges. We shall employ this characterization in estimating the total number of edges in the chordal extension resulting from our elimination ordering. This characterization is the following.

LEMMA 6.3 ([56]). *For a given elimination ordering $\alpha$, an edge $(u, v)$ is in $G_\alpha^*$ if and only if there is a path $P = \{z_0 = u, z_1, \ldots, z_p = v\}$ in $G$ such that $\alpha(z_i) \leq \alpha(u)$ and $\alpha(z_i) \leq \alpha(v)$, for each $i = 1, \ldots, p-1$.*

Using Lemma 6.3 and the structure of the separator tree, we claim the following characterization of the edges in a chordal extension given by our nested dissection ordering. This has also been shown by Gilbert and Tarjan as Lemmas 3 and 4 in their paper [26] .

LEMMA 6.4. *Let $\alpha$ be a nested dissection ordering, and $w, v \in G$ such that $\alpha(w) \leq \alpha(v)$. An edge $(w, v)$ is in $G_\alpha^*$ only if there exists an edge $(u, v) \in G$ such that $X_v$ is an ancestor of $X_w$, and $X_w$ is an ancestor of $X_u$.*

*Proof.* By Lemma 6.3, we know that if the edge $(w, v)$ exists in $G_\alpha^*$, then there is a path $P = \{w = z_0, z_1, \ldots, z_p = v\}$ from $w$ to $v$ such that all the vertices in the path are ordered before $w$ and $v$. We claim this implies that all the vertices in $P$ belong both to $T_w$ and $T_v$. We prove it by contradiction. For contradiction, let us assume that such is not the case, and that there is a vertex $z_i \in P$ such that $z_i \notin T_w$. Since $X_w$ is a node separator, any path from a vertex in $T_w$ to a vertex not in $T_w$ must contain a vertex belonging to a proper ancestor of $X_w$. But then such a vertex will be numbered higher than $w$, since the numbering is consistent with a post-ordering of the three nodes. By our assumption of the path $P$, this cannot be true. Thus each of the vertices on the path $P$ belongs to $T_w$. A similar argument shows that each of these vertices also belongs to $T_v$.

Thus $X_w$ and $X_v$ are ancestors of $X_{z_i}$ for every $0 \leq i \leq p$. In particular, $X_w$ and $X_v$ are ancestors of $X_{z_{p-1}}$, and the edge $(z_{p-1}, v)$ exists in $G$. The only way both $X_w$ and $X_v$ can be ancestors of another separator, is if one of them is an ancestor of the other. Since $\alpha(v) > \alpha(w)$, it follows that $X_w$ must be an ancestor of $X_w$. Hence the lemma holds. □

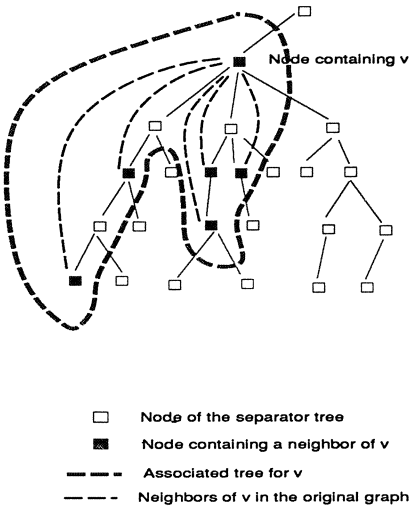


FIG. 1. *The associated tree of a node v.*

**6.3. An upper bound: Small degree graphs.** Now we shall establish an upper bound on the number of edges in the chordal graph for the ordering given by our nested dissection algorithm. We shall count the edges to a vertex $v$ from any of the vertices numbered smaller than $v$.

Let us define the *level of a node* $v$ in the tree as the distance of $v$ from the root, and denote it by $level(v)$. By a level $l$ in the tree, we refer to all the nodes at level $l$. By the level of a vertex we shall refer to the level in the separator tree of the node it belongs to. The *depth of a tree* refers to the maximum level of any node in the tree. We claim that the depth of the tree is small.

LEMMA 6.5. *The depth of the separator tree is at most* $O(\log n)$.

*Proof.* On removing a balanced separator from a graph with $n$ vertices, each of the pieces has at most $\frac{2}{3}n$ vertices. Hence the graph size decreases exponentially with the increase in recursion depth of the nested dissection algorithm. The depth of the separator tree is then at most $\log_{\frac{3}{2}} n$. □

We shall now count the number of edges to a vertex $v$ from any of the vertices numbered smaller than $v$. For that, we define the notion of an associated tree for each vertex. The *associated tree* for a vertex $v$ belonging to a separator $X$ is constructed as follows. Let $v_1, \ldots, v_k$ be the neighbors of $v$ such that $level(v_i) \geq level(v)$, for $1 \leq i \leq k$. Let $X_i$ be the separator containing $v_i$. The associated tree for $v$ is the smallest subtree rooted at $X$ containing each of the separators $X_1, \ldots, X_k$ (see Figure 1).

Lemma 6.4 implies that for every edge $(w, v) \in G^*_\alpha$ where $\alpha(v) > \alpha(w)$, $w$ must belong to the associated tree of $v$. Thus the total number of edges to $v$ from vertices

In Liu's terminology [47], the associated tree for a vertex is exactly the part of the separator tree that contains its "row subtree".

numbered lower than $v$ in the ordering is at most the number of vertices belonging to all the separators in the associated tree of $v$. We shall refer to this number for $v$ as the *cost* of $v$. Thus the total number of edges in our chordal extension is at most the sum of the costs of the vertices.

THEOREM 6.6. *The total number of edges in the chordal extension obtained by our nested dissection ordering is at most is* $O(\sqrt{d}\log^4 n)$ *times optimal, where* $d$ *is the maximum degree of the graph.*

*Proof.* Let us estimate the sum of the costs of all vertices at a given level $l_1$ in the tree. Let this level consist of separators $X_1, \ldots, X_p$. For $i = 1, \ldots, p$, consider the highest-cost vertex of $X_i$, and let $A_i$ be the associated subtree for this vertex. For each level $l \geq l_1$, let $W_l(A_i)$ be the number of vertices in $A_i$ at level $l$. Then the sum of the costs of vertices at level $l_1$ is no more than the sum, over all levels $l$ greater than $l_1$, of the value

$$\sum_{i=1}^{p} |X_i| \cdot W_l(A_i). \tag{6}$$

Let $A_i$ have $q_i$ separators $X_{i,1}, \ldots, X_{i,q_i}$ at level $l$. Since each vertex has a maximum degree of $d$, it follows that the associated tree of a vertex has at most $d$ leaves. This implies that each level of the tree has at most $d$ nodes, and hence $q_i$ is at most $d$. Substituting into (6), we get

$$\begin{aligned} \sum_{i=1}^{p}\sum_{j=1}^{q_i} |X_i||X_{i,j}| &\leq \sqrt{\sum_{i=1}^{p} q_i|X_i|^2}\sqrt{\sum_{i=1}^{p}\sum_{j=1}^{q_i}|X_{i,j}|^2} \\ &\leq \sqrt{d\sum_{i=1}^{p}|X_i|^2}\sqrt{\sum_{i=1}^{p}\sum_{j=1}^{q_i}|X_{i,j}|^2} \end{aligned} \tag{7}$$

where the first inequality follows from the Cauchy-Schwartz inequality, and the second from the fact that $q_i \leq d$. By Lemma 6.2 it follows that

$$\sqrt{d\sum_i |X_i|^2} = O(\sqrt{d\,|G^*|}\log n)$$

and similarly

$$\sqrt{\sum_{i=1}^{p}\sum_{j=1}^{q_i}|X_{i,j}|^2} = O(\sqrt{|G^*|}\log n)$$

Thus the right-hand side of (7) is $O\left(\sqrt{d}|G^*|\log^2 n\right)$. Summing over all levels $l_1$ and $l$, we conclude that there are $O(\sqrt{d}|G^*|\log^4 n)$ edges. □

Our elimination ordering hence yields a chordal graph which has only a polylog factor more edges than the optimal if the maximum degree of the graph is at most polylog in the number of nodes. This also proves that the fill for such graphs is also provably small. Moreover, many problems in practice, for example finite element problems, have small degree and thus for these problems our nested dissection ordering is guaranteed to produce near-optimal fill.

**6.4. An upper bound: Large degree graphs.** While the performance bound is polylog for small degree graphs, we cannot claim the same for the unbounded degree graphs. We can, however, claim a non-trivial performance bound which is no worse than a factor of $m^{\frac{1}{4}} \log^4 n$ times the optimal, where $m$ is the number of edges in the graph. We omit the proof for brevity. The details can be found elsewhere [1].

THEOREM 6.7. *For an unbounded degree graph $G$ with $n$ vertices and $m$ edges, the total number of edges in $G^*_\alpha$ is $O\left(|G^*|^{\frac{3}{4}}\sqrt{m}\log^{3.5} n\right)$.*

**7. Performance guarantee: Number of multiplications.** In this section, we shall establish the performance guarantee for the number of multiplications required by our nested dissection ordering. Since the cost of solving a system of linear equations is proportional to the number of multiplications required for the process, this guarantee reflects the guarantee for the total sequential time required to solve the problem using Gaussian elimination.

**7.1. A characterization of number of multiplications required.** We shall use the following characterization of the total number of multiplications required by an elimination ordering in terms of the cliques of the filled-in chordal graph. Every vertex $v$ in $G^*_\alpha$ forms a clique with all its neighbors ordered after $v$. We shall refer to this clique as the *associated clique* for the vertex, and denote it by $C_v$. The number of multiplications required to eliminate a variable $v$ is the total number of edges in the clique $C_v$. Thus the total number of multiplications required to eliminate all the variables in a chordal graph equals the sum of the number of edges in the associated cliques of each node.

**7.2. A lower bound.** Consider the case when a chordal graph has a clique of size $p$. Then for any ordering of variables in the clique, the node numbered $i$ within the clique has an associated clique of size $p - i$, for every $i$ from 1 to $p$. Thus the total number of multiplications required to eliminate all the variables in this clique is $\sum_{i=1}^{x}(p-i)^2$, which is $\Omega(p^3)$. By Lemma 5.1, since every chordal graph has a $\frac{1}{2}$-balanced clique separator, the following lemma easily follows.

LEMMA 7.1. *For any chordal graph $G^*$, if $p$ is the size of its clique separator, then $\Omega(p^3)$ is a lower bound on the number of multiplications required for any elimination ordering.*

Let $M^*$ be the least multiplication count for any elimination ordering of $G$. We shall extend Lemma 7.1 a step further to relate $M^*$ to the sizes of the separators at any level of the separator tree.

LEMMA 7.2. *Let a given level in the separator tree obtained by our algorithm have $p$ separators $X_1, \ldots, X_p$. Then $\Omega\left(\frac{\sum_{i=1}^{p}|X_i|^3}{\log^3 n}\right)$ is a lower bound on $M^*$.*

*Proof.* Let $T_i$ be the subtree rooted at $X_i$ and $G^*_i$ be the subgraph induced by the vertices of $T_i$ in $G^*$. Since $G^*_i$ is chordal by Fact 5.1, it has a clique separator. Since our separator approximation has a guarantee of $O(\log n)$, the optimal clique separator must have size $\Omega\left(\frac{|X_i|}{\log n}\right)$. By Lemma 7.1, $G^*_i$ must then require $\Omega\left(\frac{|X_i|^3}{\log^3 n}\right)$ multiplications. Since the subgraphs $G^*_1, \ldots, G^*_p$ are disjoint, it follows that any ordering in $G^*$

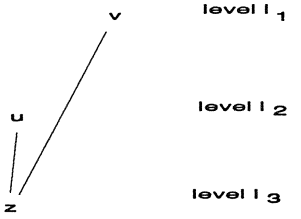


FIG. 2. *The only vertices z which can contribute to the edge $(u, v)$ must belong both to the associated tree of v and to the subtree rooted at u.*

must require $\Omega\left(\frac{\sum_{i=1}^{p}|X_i|^3}{\log^3 n}\right)$ multiplications. □

**7.3. An upper bound.** We shall now derive an upper bound on the number of multiplications required. Let $M$ be the number of multiplications required for the elimination ordering defined by the algorithm. $M$ is given by the sum over all nodes $v$ of the number of edges in $v$'s associated clique. Thus we can write $M$ as $\sum_v \sum_{e \in C_v} 1$, which is the same as $\sum_e \sum_{v: C_v \ni e} 1$. The contribution of an edge to this sum is the number of vertices containing the edge in their associated cliques. We shall refer to this quantity as the *contribution* of the edge. $M$ is hence the sum of the contributions of the edges in $G_\alpha^*$. We shall use this characterization along with Lemma 7.2 to relate $M$ to $M^*$.

THEOREM 7.3. *The number of multiplications required by our nested dissection elimination ordering is $O\left(d\log^6 n\right)$ times optimal, where d is the maximum degree of the graph.*

*Proof.* The contribution of an edge $(u, v)$ is 1 for each vertex $z$ such that $C_z$ contains the edge $(u, v)$. Without loss of generality, let us assume that $\alpha(v) > \alpha(u)$. Since $(u, v) \in G_\alpha^*$, by Lemma 6.4, $u$ must belong to the associated tree of $v$. Since $u$, $v$ and $z$ belong to the clique $C_z$, $C_z$ must contain the edges $(z, v)$ and $(z, u)$. Since $\alpha(z) < \alpha(v)$, the presence of the edge $(z, v)$ in $C_z$ implies that $z$ must also belong to the associated tree of $v$ (see Figure 2). Similarly, the fact that $(u, v)$ is in $C_z$ implies that $z$ must belong to the subtree rooted at $u$. Thus the only vertices that can contribute to the edge $(u, v)$ are those which belong to the associated tree of $v$ and also belong to the subtree rooted at $u$. Note that the latter implies that the level of such a vertex is at least as high as that of $u$.

Our approach in counting the total number of multiplications is the following. We consider all the edges in $G_\alpha^*$ that go between two given levels. For each edge we count the number of vertices in a given third level which contain the edge in its associated clique. We show that this count over all the edges between two levels is at most $O(d\log^3 nM^*)$. Since there are $O(\log^3 n)$ choices of the three levels under consideration, the total number of multiplication is $O(d\log^6 nM^*)$, and we get the theorem.

So let us consider three levels in the separator tree $l_1$, $l_2$, and $l_3$ such that $l_3 \geq l_2 \geq l_1$. Our aim is to count for each edge $(u, v)$ between a vertex $v$ in level $l_1$ and a vertex $u$ in level $l_2$, the total number of vertices in the level $l_3$ that contain $(u, v)$ in

their associated clique. Let this quantity be called $M'$. $M'$ can be written as

$$M' = \sum_{v \in \text{level } l_1} \sum_{u \in \text{level } l_2, (u,v) \in G^*_\alpha} \sum_{z \in \text{level } l_3, C_z \ni (u,v)} 1 \tag{8}$$

We want to estimate $M'$. Let us denote by $M_v$ the sum

$$\sum_{u \in \text{level } l_2, (u,v) \in G^*_\alpha} \sum_{z \in \text{level } l_3, C_z \ni (u,v)} 1$$

for a vertex $v$. Let $X_1, \ldots, X_q$ be the separators at level $l_1$, and let $v_i$ denote the vertex $v$ in $X_i$ for which $M_v$ is maximum. Then we can rewrite (8) as

$$M' = \sum_{i=1}^{q} \sum_{v \in X_i} M_v \tag{9}$$

$$\leq \sum_{i=1}^{q} \sum_{v \in X_i} M_{v_i} \tag{10}$$

$$= \sum_{i=1}^{q} |X_i| M_{v_i} \tag{11}$$

Let us now estimate the value of $M_{v_i}$. Let $A_{v_i}$ denote the associated tree of $v_i$. Let the separators in $A_{v_i}$ at level $l_2$ be $X_{i1}, \ldots, X_{iq_i}$. Each of the edges of $v_i$ to level $l_2$ must have a vertex in $A_{v_i}$ as its endpoint. Consider all the edges between $v_i$ and the vertices of the separator $X_{ij}$. There are a maximum of $|X_{ij}|$ such edges. By the above discussion, any vertex that has any of these edges in its associated clique must belong to the subtree of $A_{v_i}$ rooted at $X_i$. All such vertices at level $l_3$ must then belong to one of the separators in the subtree of $A_{v_i}$ rooted at $X_{ij}$. Let the separators in $A_{v_i}$ at level $l_3$ be $X_{ij1}, \ldots, X_{ijq_{ij}}$. Then the maximum number of vertices whose associated cliques can contain an edge between $v_i$ and a vertex in $X_{ij}$ is given by $\sum_{k=1}^{q_{ij}} |X_{ijk}|$, and there can be at most $|X_{ij}|$ such edges. Summing over all the separators in $A_{v_i}$ at level $l_2$, we get

$$M_{v_i} \leq \sum_{j=1}^{q_i} |X_{ij}| \sum_{k=1}^{q_{ij}} |X_{ijk}| \tag{12}$$

We can rewrite (11) after substituting (12) as

$$M' \leq \sum_{i=1}^{q} |X_i| \sum_{j=1}^{q_i} |X_{ij}| \sum_{k=1}^{q_{ij}} |X_{ijk}| \tag{13}$$

By using the inequality $\sum_i x_i y_i z_i \leq \sum_i (x_i^3 + y_i^3 + z_i^3)$, we can rewrite (13) as

$$\begin{aligned} M' &\leq \sum_{i=1}^{q} \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} |X_i|^3 + \sum_{i=1}^{q} \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} |X_{ij}|^3 + \sum_{i=1}^{q} \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} |X_{ijk}|^3 \\ &= \sum_{i=1}^{q} |X_i|^3 \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} 1 + \sum_{i=1}^{q} \sum_{j=1}^{q_i} |X_{ij}|^3 \sum_{k=1}^{q_{ij}} 1 + \sum_{i=1}^{q} \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} |X_{ijk}|^3 \end{aligned} \tag{14}$$

Since each vertex has degree at most $d$, it follows that the associated tree of each of the vertices has at most $d$ separators at any level. Hence we have $\sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} 1 \leq d$ for all $i$, and $\sum_{k=1}^{q_{ij}} 1 \leq d$ for all $i$ and $j$. We can then rewrite (14) as

$$M' \leq d \cdot \sum_{i=1}^{q} |X_i|^3 + d \cdot \sum_{i=1}^{q} \sum_{j=1}^{q_i} |X_{ij}|^3 + \sum_{i=1}^{q} \sum_{j=1}^{q_i} \sum_{k=1}^{q_{ij}} |X_{ijk}|^3 \tag{15}$$

Note that each of the terms on the right hand side of (15) is a sum over the (disjoint) separators at a single level, and hence we can apply Lemma 7.2. We get

$$M' \leq dO(M \log^3 n) + dO(M \log^3 n) + O(M \log^3 n) \tag{16}$$

$$= O(dM \log^3 n) \tag{17}$$

As mentioned before, the total number of multiplications is the sum of $M'$ over all the possible choices of $l_1$, $l_2$, and $l_3$. There being $O(\log^3 n)$ such possible choices, the theorem follows. □

The theorem above shows that the performance guarantee of our nested dissection algorithm is a polylog factor if the degree of the graph is small. As mentioned earlier, low degree graphs account for many of the matrices arising in practice.

**8. Performance guarantee: Elimination height.** Since problems in numerical analysis are a favorite for parallel machines, it is natural to consider how well one can perform Gaussian elimination in parallel. The amount of parallel time required by an elimination ordering can be characterized by its height. Multiple variables can be eliminated simultaneously in parallel only if the variables do not have any dependencies between them. In the graph representation, in eliminating a vertex $v$, we update all the neighbors of $v$ that are numbered higher than $v$. Hence two vertices cannot be eliminated simultaneously if they have an edge between them. If we think of each edge being directed from the vertex with a lower number to the other, then the height of an ordering $\alpha$ is the longest directed path in the chordal graph $G^*_\alpha$. Alternate characterizations of the height in terms of the elimination tree are given by Liu [47].

An elimination ordering that minimizes height does not necessarily minimize other important quantities like fill, or the multiplication count for the ordering. In fact, for the example of a simple line graph, the minimum degree heuristic is optimal in terms of fill, but has much worse height than a nested dissection ordering. Gilbert [21] has conjectured that there is an ordering that minimizes height and simultaneously approximately minimizes fill to within a constant factor. The conjecture remains unresolved.

Finding an ordering that minimizes height itself is NP-hard [53]. Hence we have to be content with finding an ordering that approximately minimizes height. It turns out that our nested dissection elimination ordering also approximately minimizes height, and thus we obtain an algorithm that simultaneously gives low fill, number of multiplications, and height. Contrary to our performance bounds for the fill and multiplication count, the guarantee for the height is independent of the degree of the input graph, and is always a $O(\log^2 n)$ factor of the optimal. We prove this result in this section.

Bodlaender et al. [2] have independently proposed an ordering scheme similar to ours that achieves approximately minimum height. The problem of finding an ordering with small height has been studied by many researchers in the past and an excellent survey can be found in the article by Heath, Ng, and Peyton (in [10]).

Since the height of an ordering is of concern when solving a system of linear equations in parallel, it will be desirable to obtain the ordering itself in parallel.

However, we do not address that issue here. Our implementation of the algorithm at present is sequential. We use the technique of Leighton and Rao [36] for finding small balanced separators in a graph, and no efficient parallel implementations are known for it. Some work has been done [32] on parallelizing the technique, but the resulting method is still not competitive. We suspect that the algorithm of Leighton and Rao cannot be parallelized efficiently. However, we hope that other techniques for finding small graph separators will be developed, which will be more amenable to parallel implementations. The issue of generating the elimination ordering itself in parallel has been studied by other researchers [10]. However, none of the previously proposed algorithms have yielded any performance guarantees.

**8.1. A lower bound.** From the discussion on the height of an elimination ordering, it follows that the height of any elimination ordering for a clique of size $m$ is $m$. That gives us the following simple lemma.

LEMMA 8.1. *For any chordal graph $G^*$, if $m$ is the size of its clique separator, then the height of any elimination ordering must be $\Omega(m)$.*

We can build on the above lemma to get the following result.

LEMMA 8.2. *Let the largest separator in the separator tree obtained by our algorithm for a graph $G$ be $X$. Then any elimination ordering for $G$ must have height $\Omega\left(\frac{|X|}{\log n}\right)$.*

*Proof.* Let $V_X$ be the set of vertices in the subtree of the separator $X$ and $G^*$ be the chordal graph with minimum height over all elimination orders. By Theorem 5.1, and the performance guarantee of our separator algorithm (see Theorem 4.1), the graph induced by $V_X$ in $G^*$ has a clique separator of size $\Omega\left(\frac{|X|}{\log n}\right)$. This clique size is a lower bound on the height of any elimination ordering by Lemma 8.1, and hence the lemma follows.

**8.2. An upper bound.** We shall now show that the height generated by our nested dissection ordering is not too much more than the optimal height.

Consider the separator tree. Let $X$ be the largest separator in the tree. Consider all the separators at each level. One variable from each of the separators can be eliminated simultaneously as there are no direct edges between the variables of different separators. Hence the number of parallel elimination steps for eliminating all the variables at a level is no more than the size of the largest separator at the level. This size is no more than $|X|$ by assumption. Since the number of levels is $O(\log n)$, the height of the ordering is at most $O(|X| \log n)$. By Lemma 8.2, the value of $|X|$ is at most $O(\log n)$ times the minimum height of any elimination ordering. It then follows that the height of our ordering is at most $O(\log^2 n)$ times the minimum height over all orderings. We have thus proved our claim of this performance guarantee in Theorem 1.4.

**9. Experimental results.** In this section we back up the theoretically provable performance of our ordering with some experimental data. We compared the quality of our results to two publicly available codes. These two codes use two different well-known heuristics. The first is the minimum-degree heuristic code by Joseph Liu [43].

The second code is the nested dissection heuristic that is implemented in SPARSPAK [19].

The minimum-degree heuristic is by far the most commonly used and acknowledged as the most effective heuristic known for finding good elimination orderings. It has a rich history. It originated from the work of Markowitz in 1957, has undergone many enhancements over the last fifteen years, and has been incorporated in many publicly available codes like MA28, YALESMP, and SPARSPAK. Much statistics regarding the performance of this heuristic and all the enhancements are also available in literature. George and Liu [18] present an excellent survey of the developments and enhancements in the minimum-degree heuristic. They suggest that a minimum-degree heuristic with certain enhancements [43] outperforms other variations of this heuristic. We obtained the latest version of the code implementing this heuristic from Joseph Liu in July 1991, and that is what we shall refer to as the minimum-degree code for the purposes of the comparison. We also wanted to compare our nested dissection ordering against an already existing one. The SPARSPAK nested dissection was an ideal choice because of its popularity.

We compared the fill, the total number of multiplications, and the height of our ordering with those obtained by the other two codes for a variety of matrices. These matrices were obtained from the Harwell-Boeing test set of sparse matrices [7, 6]. They are symmetric positive definite matrices that are derived from real applications in the industry. They have also been extensively used as a test suite by many researchers [8, 45, 40, 46, 47]. Many of the matrices that we used came from structural engineering and finite-element analysis problems.

We implemented the algorithm for finding approximate balanced node separators as described by Leighton and Rao [36]. Their algorithm consists of repeatedly applying the algorithm for finding an approximately sparsest node separator in a graph. The algorithm for finding the node separator consists of two phases. In the first phase, a uniform concurrent flow problem is solved, and in the second phase, the solution of the concurrent flow problem is rounded to produce a node separator in the graph. The first phase requires the solution of a linear program, the time complexity of which, though polynomial, was unacceptable for our purposes. We hence turned to an approximation algorithm for solving the uniform concurrent flow problem [35], which was implemented by Sarah Kang and Philip Klein [34].

We report our results below. We give the names of the matrices from the Harwell-Boeing collection, and the actual values of the three quantities of interest for the orderings. For the other two codes, we also compute the percentage difference in the values of the three quantities as compared to the values for our ordering. The number of non-zero elements in the original matrix is given in the table for reference.

Our fill is usually within $\pm$ 11% of the minimum-degree ordering. The height of our ordering is generally better than that of the minimum-degree ordering. The latter however, has better performance in terms of the number of multiplications. Compared to the SPARSPAK nested dissection ordering, our ordering seems to fare well in all the three criteria.

Though our nested dissection algorithm seems to provide competitive results, its practical use is limited due to the computationally intensive algorithm for finding the

TABLE 2
*Comparison of fill: fill is the total number of elements in the matrix that were either non-zero or became non-zero during the course of elimination*

| *matrix* | *Order* | *# entries* (symmetric) | *Our ordering* # | *Minimum Degree* # | *Minimum Degree* % Change | *SPARSPAK* # | *SPARSPAK* % Change |
|---|---|---|---|---|---|---|---|
| CANN24 | 24 | 92 | 213 | 214 | +0% | 228 | +7% |
| CANN61 | 61 | 309 | 752 | 669 | -11% | 781 | +4% |
| CANN96 | 96 | 432 | 1895 | 1856 | -2% | 2166 | +14% |
| CANN144 | 144 | 720 | 1683 | 1746 | +4% | 1812 | +8% |
| CANN187 | 187 | 839 | 3776 | 3735 | -1% | 4067 | +8% |
| CANN229 | 229 | 1033 | 5502 | 5883 | +7% | 7439 | +35% |
| BCSSTK01 | 48 | 224 | 901 | 906 | +0% | 1072 | +19% |
| BCSSTK04 | 132 | 1890 | 7121 | 6544 | -8% | 9414 | +32% |
| BCSSTK05 | 153 | 1288 | 5022 | 4524 | -10% | 5415 | +8% |
| BCSSTK06 | 420 | 4140 | 22249 | 20782 | -7% | 24116 | +8% |
| BCSPWR02 | 49 | 108 | 212 | 215 | +1% | 265 | +25% |
| BCSPWR05 | 443 | 1033 | 2720 | 2425 | -11% | 4557 | +67% |
| DWT193 | 193 | 1843 | 8556 | 8155 | -5% | 9489 | +11% |
| DWT209 | 209 | 976 | 4118 | 3812 | -7% | 6263 | +52% |
| NOS4 | 100 | 347 | 1515 | 1206 | -20% | 1754 | +16% |

approximate separators. Our algorithm may run for hours while the minimum degree heuristic algorithm or the SPARSPAK nested dissection algorithm might terminate in minutes or even seconds.

**10. Conclusions and open issues.** Our study suggests some new directions for further research and many open issues. We list them here.

- Improving the performance bounds for the ordering problems: The performance guarantees for the fill and the operation counts for our nested dissection ordering depend upon the maximum degree of the graph associated with the coefficient matrix. It is a challenging problem to find a polynomial-time ordering algorithm whose performance guarantees are independent of the degree of the input graph. A simpler problem might be to obtain an ordering algorithm whose performance guarantees are proportional to the *average* degree of the input graph. Such a result will be interesting even for the cases where the graph has excluded minors.
- Experiments with variants of our nested dissection algorithm: While our nested dissection algorithm seems to perform well in practice, we have not yet experimented with variants of our algorithm. We think that further experience with this algorithm might suggest practical enhancements to the elimination orderings produced by the algorithm. We point out again that the minimum-degree code against which we compare our heuristic has been tuned and adjusted over many years.
- Finding in parallel an elimination ordering of small height: Our nested dissection ordering is a good ordering for solving sparse linear systems in parallel. However, our algorithm for finding the elimination ordering itself is inherently sequential at present. That is because no parallel approximation algorithms are yet known for finding balanced separators in a graph. It is of interest to find a parallel algorithm that produces an ordering that has provably small height.

TABLE 3
*Comparison of multiplication count*

| *matrix* | *Order* | *# entries* (symmetric) | *Our ordering* # | *Minimum Degree* # | *Minimum Degree* % Change | *SPARSPAK* # | *SPARSPAK* % Change |
|---|---|---|---|---|---|---|---|
| CANN24 | 24 | 92 | 1076 | 895 | -17% | 1162 | +8% |
| CANN61 | 61 | 309 | 5794 | 3757 | -35% | 6201 | +7% |
| CANN96 | 96 | 432 | 22983 | 22360 | -3% | 30349 | +32% |
| CANN144 | 144 | 720 | 13165 | 14435 | +10% | 14172 | +8% |
| CANN187 | 187 | 839 | 49313 | 48754 | -1% | 58165 | +18% |
| CANN229 | 229 | 1033 | 104839 | 119890 | +14% | 184882 | +76% |
| BCSSTK01 | 48 | 224 | 8688 | 8893 | +2% | 11706 | +35% |
| BCSSTK04 | 132 | 1890 | 201202 | 144314 | -28% | 316461 | +57% |
| BCSSTK05 | 153 | 1288 | 95389 | 51549 | -46% | 110254 | +16% |
| BCSSTK06 | 420 | 4140 | 815252 | 639199 | -21% | 993936 | +22% |
| BCSPWR02 | 49 | 108 | 772 | 658 | -15% | 1147 | +48% |
| BCSPWR05 | 443 | 1033 | 17299 | 11568 | -33% | 50220 | +190% |
| DWT193 | 193 | 1843 | 229852 | 185812 | -19% | 291769 | +27% |
| DWT209 | 209 | 976 | 60381 | 45457 | -25% | 126012 | +109% |
| NOS4 | 100 | 347 | 15967 | 8585 | -46% | 19331 | +21% |

TABLE 4
*Comparison of height*

| *matrix* | *Order* | *# edges* (symmetric) | *Our ordering* # | *Minimum Degree* # | *Minimum Degree* % Change | *SPARSPAK* # | *SPARSPAK* % Change |
|---|---|---|---|---|---|---|---|
| CANN24 | 24 | 92 | 9 | 11 | +22% | 10 | +11% |
| CANN61 | 61 | 309 | 14 | 24 | +71% | 18 | +29% |
| CANN96 | 96 | 432 | 28 | 26 | -7% | 36 | +29% |
| CANN144 | 144 | 720 | 16 | 18 | +12% | 20 | +25% |
| CANN187 | 187 | 839 | 32 | 42 | +31% | 34 | +6% |
| CANN229 | 229 | 1033 | 48 | 52 | +8% | 71 | +48% |
| BCSSTK01 | 48 | 224 | 25 | 24 | -4% | 30 | +20% |
| BCSSTK04 | 132 | 1890 | 61 | 86 | +41% | 72 | +18% |
| BCSSTK05 | 153 | 1288 | 41 | 84 | +105% | 43 | +5% |
| BCSSTK06 | 420 | 4140 | 97 | 138 | +42% | 92 | -5% |
| BCSPWR02 | 49 | 108 | 5 | 13 | +160% | 9 | +80% |
| BCSPWR05 | 443 | 1033 | 23 | 31 | +35% | 56 | +143% |
| DWT193 | 193 | 1843 | 58 | 92 | +59% | 73 | +26% |
| DWT209 | 209 | 976 | 32 | 54 | +69% | 54 | +69% |
| NOS4 | 100 | 347 | 24 | 30 | +25% | 27 | +12% |

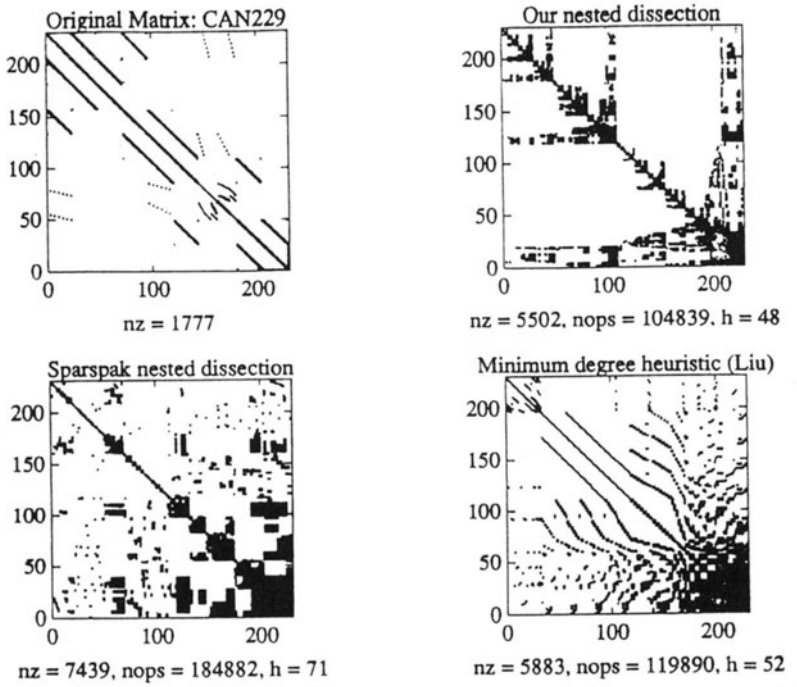


FIG. 3. *The quality of the elimination orderings produced by the three codes are compared. The original matrix from the Harwell-Boeing test-suite and is called CAN229. The fill, the number of multiplications, and the height for the orderings are specified.*

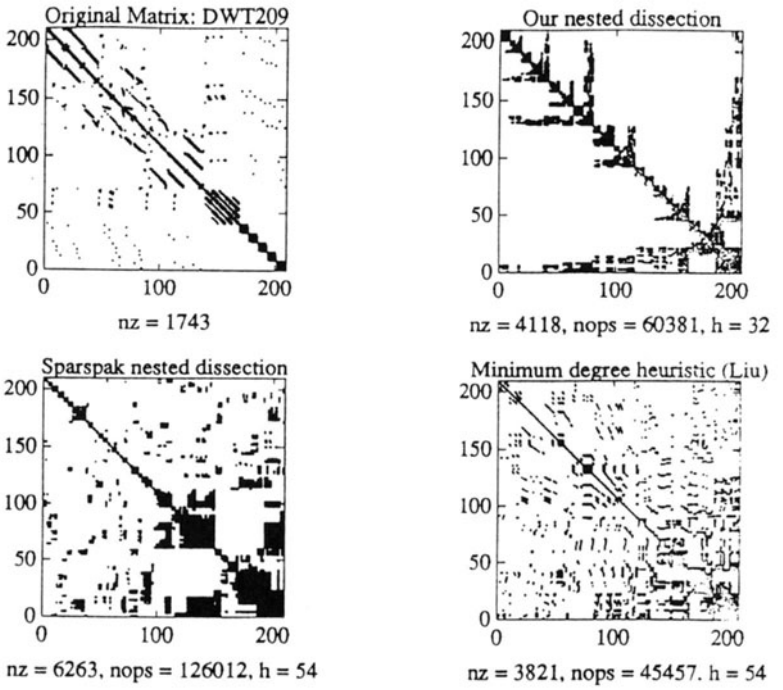


FIG. 4. *The quality of the elimination orderings produced by the three codes are compared. The original matrix from the Harwell-Boeing test-suite and is called DWT209. The fill, the number of multiplications, and the height for the orderings are specified.*

Finding a parallel algorithm for approximating minimum balanced node separators in a graph is independently of much interest.

- Running time for finding good balanced separators: The running time of our nested dissection algorithm for finding an elimination ordering directly depends upon the running time of the balanced separator algorithm. For the algorithm to gain acceptance, we must have a faster approximate separator algorithm. Separators have numerous other applications also and hence having fast separator algorithms is of much independent interest.

**11. Acknowledgments.** We gratefully acknowledge the contributions of Sarah Kang, John Gilbert, and R. Ravi to this work.

## REFERENCES

[1] A. Agrawal, "Network Design and Network Cut Dualities: Approximation Algorithms and Applications," Ph.D. thesis, Technical Report CS-91-60, Brown University (1991).

[2] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson and T. Kloks, "Approximating treewidth, pathwidth, and minimum elimination tree height," Technical Report CSL-90-01, Xerox Corporation, Palo Alto Research Center (1990).

[3] E. Cuthill, and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," *Proceedings of the 24th National Conference of the ACM* (1969), pp. 157-172.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid, "On George's nested dissection method," *SIAM Journal on Numerical Analysis, vol. 13* (1976), pp. 686-695.

[5] I. Duff, N. Gould, M. Lescrenier, and J. K. Reid, "The multifrontal method in a parallel environment," in *Advances in Numerical Computation*, M. Cox and S. Hammarling, eds., Oxford University Press (1990).

[6] I. Duff, R. Grimes, and J. G. Lewis, "Users' guide for the Harwell-Boeing sparse matrix collection," Manuscript (1988).

[7] I. Duff, R. Grimes, and J. G. Lewis, "Sparse matrix test problems," *ACM Transactions on Mathematical Software, vol. 15* (1989), pp. 1-14.

[8] I. Duff, and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations," *ACM Transactions on Mathematical Software, vol. 9* (1983), pp. 302-325.

[9] I. Duff, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press (1986).

[10] K. A. Gallivan et al. *Parallel Algorithms for Matrix Computations*, SIAM (1990).

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[12] George, J. A., "Computer implementation of a finite element method," Tech. Report STAN-CS-208, Stanford University (1971).

[13] George, J. A., "Block elimination of finite element system of equations," in *Sparse Matrices and Their Applications*, D. J. Rose and R. A. Willoughby, eds., Plenum Press (1972).

[14] George, J. A., "Nested Dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis 10* (1973), pp. 345-367.

[15] George, J. A., "An automatic one-way dissection algorithm for irregular finite-element problems," *SIAM Journal on Numerical Analysis, vol. 17* (1980), pp. 740-751.

[16] George, J. A., and J. W. Liu, "An automatic nested dissection algorithm for irregular finite-element problems," *SIAM Journal on Numerical Analysis, vol. 15* (1978), pp. 1053-1069.

[17] George, J. A., and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Inc. (1981).

[18] George, J. A., and J. W. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Review, vol. 31* (1989), pp. 1-19.

[19] George, J. A., J. W. Liu, and E. G. Ng, "User's guide for SPARSPAK: Waterloo sparse linear equations package," Tech. Rep. CS78-30 (revised), Dept. of Computer Science, Univ. of Waterloo, Waterloo, Ontario, Canada (1980).

[20] N. E. Gibbs, W. G. Poole Jr., and P. K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix," *SIAM Journal on Numerical Analysis, vol. 13* (1976), pp. 236-250.

[21] J. R. Gilbert, "Some Nested Dissection Order is Nearly Optimal," *Information Processing Letters* 26 (1987/88), pp. 325-328.

[22] J. R. Gilbert, personal communication (1989).

[23] J. R. Gilbert and H. Hafsteinsson, "Approximating treewidth, minimum front size, and minimum elimination tree height," manuscript, 1989.

[24] J. R. Gilbert, D. J. Rose and A. Edenbrandt, "A separator theorem for chordal graphs," *SIAM J. Alg. Disc. Meth. 5* (1984), pp. 306-313.

[25] J. R. Gilbert, and R. Schreiber, "Hightly parallel sparse Cholesky factorization," Tech. Report CSL-90-7, Xerox Palo Alto Research Center, 1990.

[26] J. R. Gilbert, and R. E. Tarjan, "The analysis of a nested dissection algorithm," *Numerische Mathematik, vol. 50* (1987), pp. 377-404.

[27] J. R. Gilbert, and E. Zmijewski, "A parallel graph partitioning algorithm for a message-passing multiprocessor," *International Journal of Parallel Programming, vol. 16* (1987), pp. 427-449.

[28] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).

[29] A. J. Hoffman, M. S. Martin, and D. J. Rose, "Complexity bounds for regular finite difference and finite element grids," *SIAM Journal on Numerical Analysis, vol. 10* (1973), pp. 364-369.

[30] J. Jess, and H. Kees, "A data structure for parallel L/U decomposition," *IEEE Transactions on Computers, vol. 31* (1982), pp. 231-239.

[31] U. Kjærulff, "Triangulation of graphs - Algorithms giving small total state space," *R 90-09, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg* (1990).

[32] P. N. Klein, "A parallel randomized approximation scheme for shortest paths," Technical Report CS-91-56, Brown University (1991).

[33] P. N. Klein, A. Agrawal, R. Ravi and S. Rao, "Approximation through multicommodity flow," *Proceedings of the 31st Annual IEEE Conference on Foundations of Computer Science,* (1990), pp. 726-737.

[34] P. N. Klein, and S. Kang, "Approximating concurrent flow with uniform demands and capacities: an implementation," Technical Report CS-91-58, Brown University (1991).

[35] P. Klein, C. Stein and É. Tardos, "Leighton-Rao might be practical: faster approximation algorithms for concurrent flow with uniform capacities," *Proceedings of the 22nd ACM Symposium on Theory of Computing* (1990), pp. 310-321.

[36] F. T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms," *Proceedings of the 29th Annual IEEE Conference on Foundations of Computer Science* (1988), pp. 422-431.

[37] F. T. Leighton, F. Makedon and S. Tragoudas, personal communication, 1990

[38] C. Leiserson, and J. Lewis, "Orderings for parallel sparse symmetric factorization," in *Parallel Processing for Scientific Computing*, G. Rodrigue, ed., Philadelphia, PA, 1987, SIAM, pp. 27-32.

[39] M. Leuze, "Independent set orderings for parallel matrix factorization by Gaussian elimination," *Parallel Computing, vol. 10* (1989), pp. 177-191.

[40] J. Lewis, B. Peyton, and A. Pothen, "A fast algorithm for reordering sparse matrices for parallel factorization," *SIAM Journal on Scientific and Statistical Computing, vol. 10* (1989), pp. 1156-1173.

[41] R. J. Lipton, D. J. Rose and R. E. Tarjan, "Generalized nested dissection," *SIAM Journal on Numerical Analysis 16* (1979), pp. 346-358.

[42] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem, *SIAM Journal on Computing 9* (1980), pp. 615-627.

[43] J. W. Liu, "Modification of the minimum degree algorithm by multiple elimination," *ACM Transactions on Mathematical Software, vol. 12* (1985), pp. 141-153.

[44] J. W. Liu, "Reordering sparse matrices for parallel elimination," *Parallel Computing, vol. 11* (1989), pp. 73-91.

[45] J. W. Liu, "The minimum degree ordering with constraints," *SIAM Journal on Scientific and Statistical Computing, vol. 10* (1989), pp. 1136-1145.

[46] J. W. Liu, "A graph partitioning algorithm by node separators," *ACM Transactions on Mathematical Software, vol. 15* (1989), pp. 198-219.

[47] J. W. Liu, "The role of elimination trees in sparse factorization," *SIAM Journal on Matrix Analysis and Applications, vol. 11* (1990), pp. 134-172.

[48] J. W. Liu, and A. Mirzaian, "A linear reordering algorithm for parallel pivoting of chordal graphs," *SIAM Journal on Discrete Mathematics, vol. 2* (1989), pp. 100-107.

[49] J. W. Liu, and A. H. Sherman, "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices," *SIAM Journal on Numerical Analysis, vol. 13* (1976), pp. 198-213.

[50] F. Makedon, and S. Tragoudas, "Approximating the minimum net expansion: near optimal solutions to circuit partitioning problems," Manuscript (1991).

[51] S. Parter, "The use of linear graphs in Gaussian elimination," *SIAM Review, vol. 3* (1961), pp. 364-369.

[52] F. Peters, "Parallel pivoting algorithms for sparse symmetric matrices," *Parallel Computing, vol. 1* (1984), pp. 99-110.

[53] A. Pothen, "The complexity of optimal elimination trees," Tech. Report CS-88-16, Department of Computer Science, The Pennsylvania State University, University Park, PA, 1988.

[54] D. J. Rose, "Triangulated graphs and the elimination process," *Journal of Math. Anal. Appl. 32* (1970), p. 597-609.

[55] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in *Graph Theory and Computing*, R. C. Read, ed., Academic Press (1972), pp. 183-217.

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

[57] R. Schreiber, "A new implementation of sparse Gaussian elimination," *ACM Trans. on Mathematical Software 8:3* (1982), pp. 256-276.

[58] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM J. Algebraic and Discrete Methods 2* (1981), pp. 77-79.