Parallel Gaussian Elimination with Linear Work and Fill

Claudson Bornstein Bruce Maggs Gary Miller
R. Ravi*

May 1997

CMU-CS-97-133

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

* Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213

Bruce Maggs is supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contract F196828-93-C-0193, by ARPA Contracts F33615-93-1-1330 and N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute. Gary Miller is supported in part by NSF Grant CCR-95-05472 and ARPA under Contract N00014-95-1-1246. Ravi is supported in part by an NSF CAREER Award CCR-96-25297.

Keywords: Linear fill, Height, Parallel, Interval graphs, Chordal graphs, Gaussian elimination

Abstract

This paper presents an algorithm for finding parallel elimination orderings for Gaussian elimination. Viewing a system of equations as a graph, the algorithm can be applied directly to interval graphs and chordal graphs. For general graphs, the algorithm can be used to parallelize the ordering produced by some other heuristic such as minimum degree. In this case, the algorithm is applied to the chordal completion that the heuristic generates from the input graph. In general, the input to the algorithm is a chordal graph G with n nodes and m edges. The algorithm produces an ordering with height at most $O(\log^3 n)$ times optimal, fill at most $O(m)$, and work at most $O(W^*(G))$, where $W^*(G)$ is the minimum possible work over all elimination orderings for G. Experimental results show that when applied after some other heuristic, the increase in work and fill is usually small. In some instances the algorithm obtains an ordering that is actually better, in terms of work and fill, than the original one. We also present an algorithm that produces an ordering with a factor of $\log n$ less height, but with a factor of $O(\sqrt{\log n})$ more fill.

1 Introduction

One of the most popular methods for solving a system of linear equations is *Gaussian elimination.* The crux of this method is to pivot on the variables of the system one-at-a-time according to some ordering. For example, suppose that the variables $x_1, \ldots, x_n$ are to be eliminated according to an ordering π. Then in the ith pivoting step, variable $x_{\pi(i)}$ is eliminated from equations $\pi(i+1), \pi(i+2), \ldots, \pi(n)$.

The system of equations is typically represented as a matrix, and as the pivots are performed some entries in the matrix that were originally zero may become non-zero. The number of new non-zeros produced in solving the system is called the *fill.* Among the many different orderings of the variables, one is typically chosen so as to minimize the fill. Minimizing the fill is desirable because it limits the amount of storage needed to solve the problem, and also because the fill is strongly correlated with the total number of operations (work) performed.

Gaussian elimination can also be viewed as an algorithm that is performed on the graph whose adjacency matrix is the matrix representing the system of equations [29, 31]. Pivoting on a variable corresponds to removing a vertex from the graph and forming a clique of its neighbors. The number of new edges added to the graph in this process constitutes the fill. Throughout this paper we assume that our matrices are symmetric positive definite, so that our graphs are undirected, our pivots are always non-zero, and we can ignore the issue of numerical stability.

1.1 Heuristics for sparse Gaussian elimination

A number of heuristics for minimizing fill for sparse matrices are available, the most popular being nested dissection and minimum-degree.

Nested dissection, as the name suggests, is a recursive elimination procedure. It identifies a balanced separator in the graph and sets the nodes in the separator apart for elimination at the very end. The components resulting from removing the separator are recursively ordered, one after the other, and placed before the separator in the elimination ordering. George [12] first proposed this method for eliminating nodes in a mesh, and later generalized it in a paper with Liu [13] for eliminating the nodes in an arbitrary graph. Bounds on the fill produced by nested dissection orderings are known for planar graphs and arbitrary graphs with bounded degree [1, 14, 21].

The *minimum-degree* heuristic repeatedly finds a vertex of minimum degree and eliminates it. This heuristic originated with the work of Markowitz in the late 50's and has undergone several enhancements in the years since [23]. Its popularity is attested to by its inclusion in various publicly available codes such as MA28, YALESMP, and SPARSPAK. In contrast to nested dissection, no performance guarantee is known for the fill produced by this heuristic. In fact, there exist graphs for which the fill induced by the minimum-degree ordering can be very high [4].

Recently, some hybrid algorithms have been shown to experimentally produce fill that compares favorably with those produced by either minimum-degree or nested dissection alone. Hendrickson and Rothberg [15], and independently Liu and Ashcraft [2], proposed algorithms that first find separators that partition the graph into small components. The minimum-degree heuristic is then used to order the vertices within each component, and also within the separators. In practice, both algorithms produce orderings that compare favorably with state-of-the-art minimum-degree and nested-dissection algorithms, but no bounds on the amount of fill that they introduce are known.

1.2 Our results

In this paper we focus on elimination orderings for chordal graphs. Chordal graphs are a natural choice, because they are rich in structure and because they are intimately related to Gaussian elimination orderings; in fact, chordal graphs are exactly the class of graphs that have zero-fill elimination orderings. Moreover, since any elimination ordering constructs a chordal completion of a graph, that is, adds edges to the graph so as to make it chordal, we can apply our algorithm to the chordal completion produced by any good ordering algorithm.

Chordal graphs already have zero-fill elimination orderings, so how can we possibly improve on that? We propose that some extra fill might be tolerable, if parallelism can be exposed. Although we have thus far described Gaussian elimination as if vertices were eliminated one-at-a-time, in fact a set of vertices can be eliminated in parallel if they are independent, i.e., no two vertices in the set are neighbors. Thus, in a parallel elimination ordering, we allow independent sets to be eliminated in one step, and we define the *height* to be the total number of steps. A lower bound on the height of any elimination ordering is the size of the maximum clique in the graph, since the vertices in a clique cannot be eliminated in parallel.

Although a strictly sequential ordering has height n, it is often possible to expose some parallelism in a sequential ordering. The idea is to view a sequential ordering as a partial ordering that constrains each vertex to be eliminated before any of its neighbors that appear later in the sequential ordering. Thus, we can define the height of a sequential ordering to be the minimum-height parallel ordering that is consistent with the partial order. Nested dissection is known to produce low-height orderings, in particular, within a polylogarithmic factor of the minimum possible [1, 28]. On the other hand, minimum-degree orderings can have a polynomial factor more height than the minimum possible (e.g., a path).

Trying to achieve fast parallel solutions while keeping the space overhead minimal corresponds to finding an ordering that has simultaneously low height and low fill. Gilbert conjectured the existence of a parallel elimination ordering that has the minimum possible height among all orderings and fill that is only a constant factor more than the number of edges in a minimum-fill ordering (see [3]). The hope was that a small increase in fill could be traded for faster parallel solutions. Aspvall [3] disproved this conjecture, however, by exhibiting a graph for which any ordering that has the minimum possible height requires a polynomial factor more fill than the minimum possible.

Given an interval graph (a subclass of chordal graphs) with n vertices and m edges, can we find an ordering with $O(m)$ fill and height close to the minimum possible? In particular, does a nested dissection ordering accomplish this? We show that the classical nested dissection procedure applied to interval graphs produces an ordering with $O(\sqrt{\log n} \cdot m)$ fill and with height at most $O(\log n)$ times the optimum for that graph. In fact, the bound on fill is tight if the nested dissection algorithm is forced to choose a $(1/2)$-balanced separator, thus providing a negative answer to our question. Even in this very restricted class of graphs, nested dissection may generate undesirable fill.

On the positive side, we show that an extension of nested dissection provides good orderings. We exhibit an interval graph algorithm that produces orderings with $O(m)$ fill and height within a factor of $O(\log^2 n)$ times the optimum. This same algorithm can then be generalized to chordal graphs, and produces orderings that also have $O(m)$ fill, while having height within an $O(\log^3 n)$ factor of the minimum possible. While this guarantee is worse in terms of height than the one nested dissection provides, it is significantly better than that given by either a perfect elimination ordering or by the minimum-degree heuristic. In addition to the balanced separators used in nested dissection, we utilize another kind of separator that we call *sentinels*. Sentinels help localize the fill produced by our orderings without increasing the height by much. In addition to the bound on the fill, we also show that the total work as well as the front size[1] of our orderings are within a constant factor of the minimum possible.

Preliminary experiments show that the overheads in fill and height are much better than predicted by our theoretical analysis. For instance, we observed the following interesting behavior in two-dimensional grids. Minimum-degree performs better than nested dissection in terms of fill and work on grids with high aspect ratio [2]. However, the orderings produced by minimum-degree are very sequential in nature and exhibit large height. Our algorithm, applied to the chordal completion obtained from the minimum-degree ordering generates an ordering that has good height and low fill and work. In fact, when compared to a nested dissection ordering of the original grid, our ordering exhibits worse height, but slightly better fill and work (See Table 2).

[1]The front size corresponds roughly to the size of a maximum clique in the filled graph.

1.3 Related work: Fill

The problem of finding an elimination ordering that minimizes the fill for arbitrary graphs is known to be NP-hard [32].

The first analysis for a variant of nested dissection for graphs with small separators (of size $O(\sqrt{n})$ in an n-node graph) was given by Lipton, Rose and Tarjan [21]. The fill introduced by this variant is $O(n \log n)$ on an n-node graph. Subsequently, Gilbert and Tarjan [14] analyzed the original nested dissection algorithm of George and Liu for planar graphs, and showed that using small separators in the recursive procedure yields a fill of $O(n \log n)$ [22]. They also point out that this method does not work in general for graphs with small separators by constructing a counterexample. Both of these papers [14, 21] also show a bound of $O(n^{\frac{3}{2}})$ on the work of the orderings. It is interesting to note that there are n-node planar graphs (square grids in particular) for which any elimination ordering introduces fill $\Theta(n \log n)$ [7].

Agrawal, Klein and Ravi [1] gave the first approximation algorithms for elimination orders that simultaneously minimize the fill, height and the work, all within a polylogarithmic factor when the degree of the input graph is bounded. Their algorithm is essentially the nested dissection algorithm using approximately minimum-size balanced node separators [18] to construct the recursive decomposition. They also analyze the fill and the height of their ordering when the degree of the graph is not bounded. The key difference between our work and these results is that we start with a chordal completion of a graph, and focus our efforts on finding parallel elimination orders with linear fill.

1.4 Related work: Height

Ignoring fill, computing an elimination ordering for a given graph with minimum height is NP-hard [30], and remains so even if an additive error in the estimate of the height is allowed [5]. Pan and Reif give one of the first analyses of the parallel height of nested dissection orderings as well as how nested dissection can be used for solving the shortest path problem in graphs [28, 27]. Bodlaender et al. [5] uses an approach similar to [1] to find elimination orders with bounds on the height and several related parameters. Both these papers [1, 5] give elimination orders with height at most $O(\log^2 n)$ times the minimum possible, for any n-node graph. Numerous heuristics without performance guarantees are also known for this problem [10, 16, 19, 20, 24, 25].

1.5 Outline

The remainder of this paper is organized as follows. In the next section, we introduce some definitions. We present two algorithms for finding parallel elimination orders for interval graphs. The first algorithm, which is based on nested dissection, is described in Section 3. The second, which has linear fill, is presented in Section 4. We then show in Section 5 how these algorithms can be used to find elimination orders for chordal graphs. Some experimental results obtained for an implementation of the algorithm in Section 5 can be found in Section 6. We conclude with some remarks in Section 7.

2 Definitions

In order to proceed, we need to establish some notation concerning matrices and graphs.

Each step of Gaussian elimination on a symmetric matrix M corresponds to choosing a vertex v in G, adding edges to G if necessary to make v's neighborhood a clique and then removing v from G. v is said to have been *eliminated* from G. Any new edges introduced by the elimination of a vertex are called *fill* edges, or simply *fill*. A vertex v is *simplicial* in G if its neighborhood $N(v)$ is a clique of G. Simplicial vertices are of special interest, since the elimination of a simplicial vertex does not introduce any fill edges.

Alternatively, we can think of Gaussian elimination as simply inserting the fill edges in a graph. In this case, the elimination of a vertex corresponds to the introduction of edges between any pair of its neighbors that are not connected, and are later in the elimination ordering than the vertex being considered. The graph augmented with all the fill edges is referred to as the *updated graph.* Given two non-adjacent vertices v and w in a graph G, there exists a fill edge (v, w) between them in the updated graph iff there exists a path from v to w going only through vertices numbered lower than, i.e., that are eliminated before, both v and w.

An ordering $v_1, v_2, \ldots, v_n$ of the vertices of G is a *perfect elimination ordering* if it does not introduce any fill edges, i.e., if each v_i is simplicial in $G - \{v_1, \ldots, v_{i-1}\}$. A graph is said to be *chordal* if it has a perfect elimination ordering. Equivalently, a graph is chordal if every simple cycle with more than three vertices has a chord [9, 31], i.e., no induced subgraph of G is isomorphic to a cycle with more than three vertices.

The *intersection graph* of a family F of sets S_i is the graph obtained by associating a vertex v_i with each set S_i, and edges (v_i, v_j) whenever S_i intersects S_j. One characterization of chordal graphs that has proved particularly useful is as the intersection graphs of subtrees, that is, connected subgraphs, of a tree. We call the tree in question a *skeleton* of the chordal graph G. Along with the subtrees it forms a *tree representation* of G. A tree representation of a graph G is said to be *minimal* if the associated skeleton has the minimum number of nodes possible. Gavril [11] and Buneman [6] showed that in a minimal representation there is a one-to-one correspondence between vertices of T and maximal cliques of G. Alternatively, we can consider the nodes of T to be formed by sets of vertices of G so that for each vertex v of G a subtree T_v induced in T by the nodes that contain v can be used to represent v. T_v is said to be the *representative subtree* of v. A minimal tree representation of G is called a *clique tree* of G.

Throughout this paper, we refer to vertices in a graph, but we reserve the term *node* to refer to vertices in the skeleton of a chordal graph and to vertices in separator trees, that is, trees whose nodes correspond to separators in the original graph. In both cases, nodes typically correspond to sets of one or more vertices. Similarly, we reserve the term *link* to refer to edges between nodes in a skeleton of a chordal graph. A subtree T_v is said to *cover* a node/link of the skeleton if that node/link is in T_v. A *terminal branch* of T is a maximal path from a leaf v to a node w in T that, except for v and w, only contains degree-2 nodes.

An important subclass of chordal graphs are the *interval graphs*, which are chordal graphs that have a skeleton that is a path. A tree representation with a path for a skeleton is also called an *interval representation*, for the representative subtrees are also just paths, and can be interpreted as intervals.

3 Parallel elimination orders for interval graphs: nested dissection

In this section we analyze a simple nested dissection algorithm that chooses a balanced separator at each step, thus producing a logarithmic depth separator tree. We show an upper bound of $O(m \cdot \sqrt{\log n})$ on the amount of fill for the orderings produced.

Given a graph G, an *α-balanced separator* of $G = (V, E)$ is a set of nodes $S \subset V$ such that no connected component of $V - S$ has more than $\alpha \cdot |V|$ vertices, for some constant fraction $\alpha < 1$. An *α-balanced separator tree* is one whose nodes are α-balanced separators of the subgraphs of G. The root of the tree is an α-balanced separator of G, and we build a tree recursively for each component and attach them as subtrees of the root. From now on, whenever we use the term *balanced separator* we simply mean an α-balanced separator, for some constant α.

Nested dissection on an interval graph I builds a balanced separator tree whose nodes are minimal separators of subgraphs of I. For interval graphs, every minimal separator of the graph corresponds exactly to the set of vertices that cover some link of its skeleton P. We order this tree so that an in-order transversal of the separator tree corresponds to a left-to-right transversal of the links of P. When necessary we will refer to an *ordered* separator tree to make it clear that we are considering a separator tree whose children are ordered as described here.

As long as the algorithm chooses α-balanced separators, the depth of the separator tree is $O(\log n)$, since each node has left and right subtrees, which have no more than an α-fraction of the vertices in the subtree rooted at that node.

3.1 Analysis

Even though the separator tree itself has depth $O(\log n)$, the corresponding elimination order can potentially be fairly unbalanced, since the separators will probably have different sizes. Every separator is a clique in the corresponding graph and hence its size is a lower bound on the height of any elimination order. If the depth of the separator tree is d, we get the following lemma, which assures that this unbalance can be at most a logarithmic factor.

Lemma 1 *[1] Let G be a graph. A depth d minimal balanced separator tree for G produces an ordering of depth within a factor of d of the optimal.*

Proof. Let s be the number of vertices in the largest minimal separator S in the tree. The ordering defined by the tree has depth at most $d \cdot s$. Let G' be the graph obtained from G by adding all the fill edges introduced by an optimal ordering of the vertices of G. G' is chordal since that optimal ordering for the vertices of G is a perfect elimination ordering for G'. G is a subgraph of G' and the largest minimal separator of G' must have at least s vertices. Since G' is chordal, every minimal separator of G' is a clique and thus any elimination ordering for the vertices of G' must require at least s steps. ∎

We now proceed to bound the total number of fill edges introduced by the nested dissection algorithm. In the lemmas that follow, we only consider non-trivial interval graphs, that is, we assume that the graphs in question have at least two distinct maximal cliques. We also assume that any pre-existing simplicial vertices have been eliminated from the graph. Thus, no representative subtree consists of a single node since we do not obtain a new interval representation after the initial elimination of simplicial vertices.

Figure 1 shows part of a separator tree of an interval graph. Each node in the tree corresponds to a minimal separator of the graph. The elimination ordering specified by the separator tree might introduce fill edges between a vertex v in A and vertices in B's right subtree, as depicted by the dotted edges. In this case v must be adjacent to some vertex in B's right subtree as depicted by the edge in the figure.

We define an *inner path* of a node A in an ordered binary tree as the path that starts with the edge to the left or right child of A, and goes all the way to the in-order predecessor or successor of A, respectively. The next lemma states that amounts of fill in excess of $O(m)$ must be between a node and its inner paths.

Lemma 2 *Let I be a connected interval graph. The total amount of fill between vertices in separator nodes of an ordered balanced separator tree of I and vertices not in the corresponding inner paths is $O(m)$.*

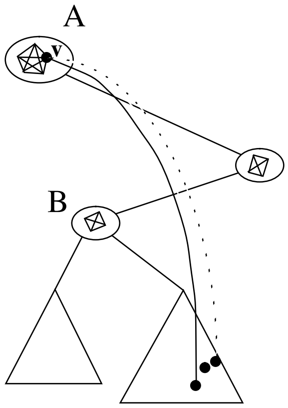

Figure 1: fill among vertices in a separator tree of an interval graph.

Proof. Let v be a vertex in a separator node A, which has fill to vertices not in one of A's inner paths. Let's examine the right subtree of A. The left one is analogous. Let B be a node in A's right inner path, whose right subtree contains vertices to which v has fill. Then there exists a path from v to vertices in separator nodes in B's right subtree that does not go through vertices in B. Otherwise there would be no fill to vertices in that subtree, since the vertices in B are eliminated after those in the subtree. Thus, v must cover the link in the skeleton corresponding to B, and the links corresponding to each of the nodes in B's left subtree. Hence v is adjacent to all vertices in B and in B's left subtree. In particular, this implies that for a given v, only one such B can exist. Otherwise, let C be another such node in the right inner path of A. Without loss of generality, C is in B's left subtree, and thus v is already adjacent to all vertices in C's left and right subtrees, which contradicts the assumption that there is fill from v to a vertex in C's right subtree.

B was chosen so that its right subtree has at most an α-fraction of the vertices in the whole subtree rooted at B. Moreover, v is adjacent to all vertices in B and in B's left subtree. Thus the number of fill edges from v to the right subtree is $O(d(v))$. By applying this same argument to all vertices to account for fill to vertices not in their inner paths of the corresponding separators we get a total of $O(m)$ fill edges. ■

Lemma 2 allows us to concentrate on fill involving vertices in inner paths. Consider one such inner path.

Lemma 3 *Let I be a connected interval graph, and let V_0 be a node in an ordered balanced separator tree of I. Let $V_1, V_2, \ldots, V_k$ be the nodes in V_0's right inner path. The total amount of fill between V_0 and vertices in its right inner* *path is at most* $O(\sqrt{k} \cdot \sum_{i=0}^{k} |V_i|^2)$.

Proof. Let $n_i = |V_i|$. The total amount of fill from V_0 to its right inner path is at most $n_0 \cdot (n_1 + \ldots + n_k)$. Let $n_1 + n_2 + \ldots + n_k = d$. We must bound the amount of fill as a function of the sum of the n_i's squared, which is a lower bound on the number of edges in $\cup V_i$. Thus, we are looking for the least number x such that $n_0 \cdot d \leq x \cdot (\sum_{i=0}^{k} n_i^2)$. But $\sum_{i=1}^{k} n_i^2 \geq k(d/k)^2$. Thus we obtain $x \cdot n_0^2 - d \cdot n_0 + x \cdot d^2/k \geq 0$. As long as x is positive, it suffices to choose x such that this second degree equation on n_0 does not have two distinct real roots. Thus x must satisfy $d^2 - 4x^2 d^2/k \leq 0$, i.e., $x \geq \sqrt{k}/2$ and the total amount of fill is $O(\sqrt{k} \cdot \sum_{i=0}^{k} n_i^2)$. ■

Since a separator is in at most 4 inner paths, and a balanced separator tree has $O(\log n)$ depth, Lemmas 2 and 3 give the following corollary:

Corollary 1 *Let I be a connected interval graph, with a balanced separator tree. The total amount of fill induced by the ordering specified by the tree is* $O(m \cdot \sqrt{\log n})$.

Proof. According to Lemma 2, the total amount of fill is at most $O(m)$ plus the amount of fill to inner paths. Apply Lemma 3 to all inner paths. Each separator is in at most two inner paths, and has at most one left and one right inner path. Since the tree is balanced, the largest inner path has $O(\log n)$ length. All that remains to be shown is that the sum of the squares of the sizes of the separators in the tree is $O(m)$. But each separator is a clique, and each vertex shows up in only one separator, so that a separator with n_i vertices has $n_i(n_i - 1)/2$ edges that do not appear in any other separator. For $n_i > 1$, $n_i^2 \leq 2(n_i(n_i - 1))$. To handle the case $n_i = 1$ we note that the graph is connected and thus every vertex must be adjacent to at least one other vertex in the graph; since that edge can only be considered by those two vertices, we have the desired result. ■

3.2 An example

It is not hard to find examples that show that if the nested dissection algorithm is forced to choose (1/2)-balanced separators then the bound derived in the previous section is tight.

Since the algorithm being used has to choose a (1/2)-balanced minimal separator, we can build an interval graph with not too many edges and such that at each step the algorithm can only make a single choice. For simplicity we construct a graph such that the fill from the root node to one of its inner paths is enough to achieve the bound.

Except for the root separator, and the separators in the root's right inner path, all other separators have a single vertex, and are distributed so as to make sure that all separators are in fact (1/2)-balanced separators for the subgraphs induced by the subtree rooted at that separator node. Let n_0 be the size of the root separator, and let s be the size of each of the other separators in the root's right inner path. Our sample graph can be obtained by making a clique out of each of the separators, and connecting vertices in each separator with those in its in-order successor and predecessor separator nodes.

The total number of vertices in the graph is $O(2^k \cdot s + n_0)$, where k is the number of nodes other than the root separator in the root's right inner path.

If $k = \log n_0$ then the total number of vertices in the graph is $O(n_0 \cdot s)$.

The vertices in the root separator have a total of $O(n_0^2 + n_0 \cdot s)$ edges, $O(n_0^2)$ to themselves, and $O(n_0 \cdot s)$ to the vertices in a separator with s vertices that they are adjacent to. The vertices in that separator of size s have $O(s \cdot (n_0 + s))$ edges, while the remaining separators with s vertices have $O(s^2)$ edges. One other vertex is adjacent to the n_0 vertices in the root separator thus having $O(n_0)$ edges, and $O(2k)$ vertices are adjacent to one of the separators with s vertices, and thus have $O(s)$ edges each. The remaining $O(n_0 \cdot s)$ vertices have $O(1)$ edges each. The total number of edges in the graph is thus $O(k \cdot s^2 + n_0^2)$.

The total amount of fill is $\Omega(n_0 \cdot s \cdot \log n_0)$ and if we choose $s = n_0/\sqrt{\log n_0}$, the total number of edges is $O(n_0^2)$, while the amount of fill is $\Omega(n_0^2 \cdot \sqrt{\log n_0})$.

4 A linear-fill $O(\log^2 n)$-depth algorithm

In this section we present a recursive algorithm that, given an interval graph, finds an $O(\log^2 n)$-depth separator tree that represents the elimination ordering for the vertices in the graph. Unlike traditional separator trees, vertices can appear multiple times in the tree.

The algorithm is composed of three phases, which operate on a skeleton path of an interval graph. The analysis of the algorithm uses a potential function whose value is $O(m)$ initially and is used to account for the fill edges. The last of the three phases is carried out to ensure that we do not charge to the same part of the potential function multiple times. This is done by keeping track of which edges are "depleted" of their contribution to the potential, and dividing the graph into subgraphs with predominantly non-depleted edges to recurse on.

The first phase, *homogenize*, finds up to $k = O(\log n)$ separators that divide the graph into $k+1$ components. The sizes of these separators are geometrically decreasing, a property that is useful in accounting for fill induced to any of these separator vertices. The next phase, which we call *halving*, is analogous to a regular nested dissection iteration: we simply select a separator that divides the skeleton of the interval graph we are currently working with in half. As in nested dissection, this ensures that the algorithm finishes within $O(\log n)$ iterations of the phases, thus producing an ordering with good height. Finally, the algorithm performs the *kill* phase. As mentioned earlier, the purpose of this phase is to ensure that the potential function is used correctly to account for the fill. The kill phase accomplishes this by choosing special kinds of separators called sentinels that localize subsequent fill to subgraphs that contain enough non-depleted edges (whose potential contribution is as yet unused). In choosing sentinels to ensure this, we incur an extra $\log n$ factor in the height of our ordering, since we select up to $\log n$ sentinels per kill phase.

We also analyze a chordal graph version of our interval graph algorithm, which is described in more detail in Section 5. The algorithms are essentially the same, except that when dealing with chordal graphs we have a skeleton that is a tree, not a path. We can apply the interval graph algorithms to eliminate each of the branches (paths) of the skeleton that lead to leaves of the tree. We repeat this step until the whole tree has been considered.

4.1 Definitions used in the algorithm

An edge is said to be an *extremity* of a tree if it is an edge to some leaf of the tree. To *insert* a link e of the skeleton into the separator tree consists of adding to the tree the separator formed by those vertices of the chordal graph whose representative subtrees cover e. A link of the skeleton is said to be a *root* if it has been inserted into the separator tree. A rooted skeleton is a skeleton whose extremities are roots.

A vertex v and its representative subtree T_v are said to be *pinned* at a link r if r is a root and either T_v covers r or T_v is a singleton covering one of the vertices of r. Some subset of the roots is said to be *depleted* in G. A vertex is said to be depleted if it is pinned at a depleted link. Whenever we refer to a pinned vertex, it is not depleted, unless explicitly stated. The term depleted is used to help us keep track of which edges in the input graph have been used to pay for fill and can no longer be used. Edges between pairs of depleted vertices are also said to be depleted. Edges between pinned vertices are said to be *pinned edges*. Unless otherwise stated, the term pinned edges only refers to non-depleted pinned edges. A vertex is said to be internal to a graph if it is not pinned. Edges to internal vertices are also said to be internal, regardless of whether the other endpoint of the edge is internal.

Let G be an interval graph, with skeleton T. We denote $P_{l,r}$ the path between and including the two links l and r in T. We denote $P(l,r)$ the interval graph obtained by restricting G to $P_{l,r}$ and eliminating any single vertex representative subtrees. Given a representative subtree T_v associated with a vertex v of G, let T'_v be the path induced in T_v by $P_{l,r}$. $P(l,r)$ is the interval graph that has those T'_v with two or more nodes in their representative subtrees on the skeleton $P_{l,r}$.

The *ply* p_e of a link e of T is the number of subtrees T_v that cover that link.

4.2 The algorithm

Given an interval graph I, and an interval representation of I, with skeleton P, whose extremities are l and r, we remove any simplicial vertices of I, insert both l and r into the separator tree, and apply the procedure homogenize to it. Eliminating simplicial vertices does not change the skeleton of the graph, but rather eliminates any single node representative subtrees. Since none of the steps of the algorithm creates these single-node sub-paths, later steps do not have to deal with them.

Each step of the algorithm inserts one or more links of the skeleton into the separator tree, resulting in a number of subgraphs. Let $K_l(e_i, e_{i+1})$ denote the graph obtained from $P(e_i, e_{i+1})$ by removing all vertices pinned at l as well as those that cover both e_i and e_{i+1}.

The algorithm consists of three major procedures that call each other recursively. Each procedure inserts some vertices into the separator tree thus creating subgraphs to which the next procedure is applied as long as there exists at least one vertex internal to the subgraph. Each procedure operates on an interval graph I, a skeleton path P of I, and P's two extremities, l and r. The first procedure, homogenize, divides the interval graph in a number of subgraphs and then applies the halving procedure to each one of them. Figure 2 illustrates this process.

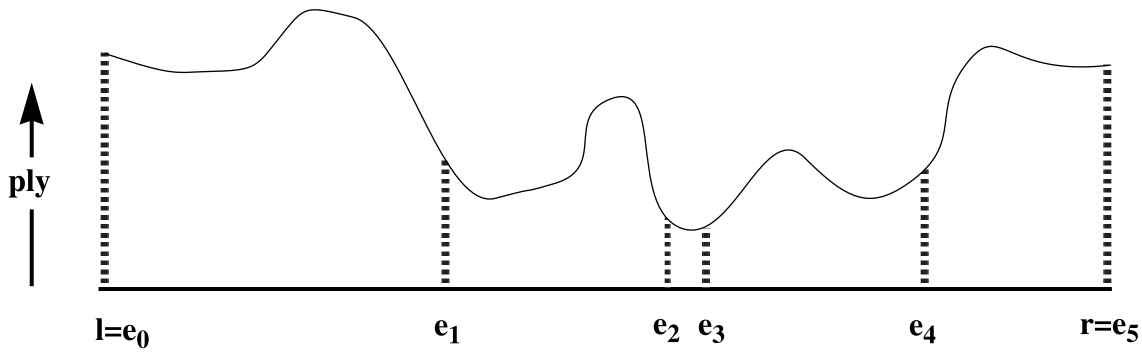

Figure 2: The homogenize procedure devides a graph into a number of subgraphs, by selecting up to $O(\log n)$ edges. Edges are chosen starting from both extremities, so that their ply decreases geometrically.

Homogenize(I, P, l, r)

Select l, and transverse P towards r, selecting the first link whose ply is at most half the ply of the last selected link. Repeat until r is reached. Apply the same algorithm from r to l. Let e_i, $0 \leq i \leq k+1$, be the links that were selected ordered from l to r, including $l = e_0$ and $r = e_{k+1}$. Insert all the selected links except l and r into the separator tree, and do Halving$(P(e_i, e_{i+1}), P_{e_i, e_{i+1}}, e_i, e_{i+1})$, for all i between 0 and k for which some vertex in $P(e_i, e_{i+1})$ is not in the elimination tree yet.

The halving procedure simply does what its name suggests, that is, divides the problem in two subproblems that are no more than half the size of the original one.

Halving(I, P, l, r)

Choose a link m in P such that when m is inserted into the separator tree, both $P_{l,m}$ and $P_{m,r}$ have no more than half as many skeleton links as P did. Then do Kill$(P(l,m), P_{l,m}, l, m)$ (and Kill$(P(m,r), P_{m,r}, m, r)$), as long as $P(l,m)$ ($P(m,r)$) has some vertex that has not been inserted in the separator tree.

Finally, the kill procedure divides an interval graph into a number of subgraphs, and from each one it removes vertices that have either been depleted or cover the entire piece of the skeleton corresponding to that subgraph. This reestablishes the preconditions of homogenize, which is applied to each of the resulting subgraphs. Figure 3 illustrates the kill procedure applied to an interval graph. The dotted paths are pinned at the depleted root l and correspond to the depleted vertices.

Kill(I, P, l, r)

Assume the ply at l is at least the ply at r, i.e., $p_l \geq p_r$. Otherwise, swap the roles of l and r in this algorithm. From r to l in P, select the first link that covers a vertex that is pinned at l. Keep scanning P towards l, and selecting the first link that covers at least twice as many vertices pinned at l as did the last selected link. Call these *milestone* links. Also select the links adjacent to the milestones, which are closer to r than the corresponding milestone, and call those *sentinels*. Insert the links that were selected into the separator tree in order, from l to r. Call those links e_i, including $l = e_0$ and $r = e_{k+1}$. Apply Homogenize$(K_l(e_i, e_{i+1}), P_{e_i, e_{i+1}}, e_i, e_{i+1})$, for all i such that $K_l(e_i, e_{i+1})$ has at least one vertex that hasn't been inserted into the separator tree. Note that when e_i and e_{i+1} are a milestone and its corresponding sentinel all vertices in $P_(e_i, e_{i+1})$ have already been ordered.

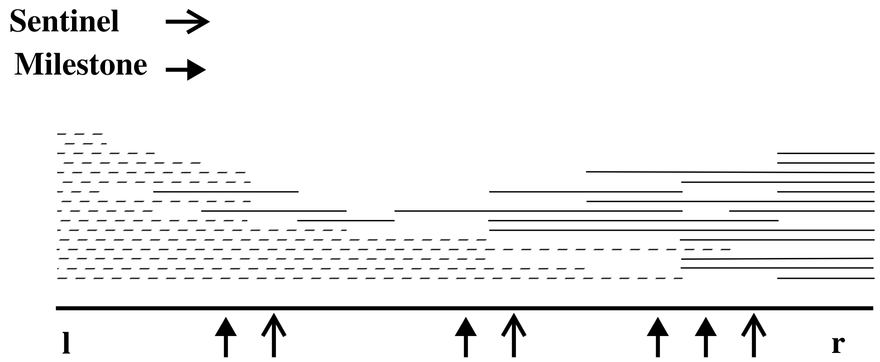

Figure 3: The kill procedure selects milestones and the corresponding sentinels according to the number of paths pinned at the depleted root l.

4.3 Analysis

A chordal graph, and in particular an interval graph I has at most n maximal cliques, and thus at most n nodes in its clique tree. Just before the first recursive invocation of homogenize, the three procedures have partitioned I into subgraphs that have skeletons with no more than half the number of links in the skeleton of I. Since the three procedures add $O(\log n)$ nodes to the separator tree, and $\log n$ iterations of those three procedures might be necessary, the depth of the tree generated by this algorithm is $O(\log^2 n)$. Since every separator is a clique of I, the ordering produced has height at most $O(C \cdot \log^2 n)$, where C is the size of the maximum clique of I.

4.3.1 Bounding the amount of fill

We can obtain bounds on the amount of fill and work that are incurred by the orderings produced by our algorithm which are summarized in the next lemma.

Lemma 4 *Let I be an interval graph with n vertices and m edges. When applied to I, the algorithm presented in Section 4.2 produces an ordering with height $O(C \cdot \log^2 n)$, fill $O(m)$, and work $O(W^*(I))$, where C is the size of the largest clique in I and $W^*(I)$ is the minimum amount of work required to perform Gaussian elimination on I according to any ordering.*

We now define a potential function that will be used to bound the amount of fill introduced by the algorithms. Let G be a chordal graph and let T be a skeleton of G at some stage in the algorithm. Let s be the number of links of

T that are not roots. Let x be the number of internal edges of G, and y the number of pinned, non-depleted edges. The potential $\phi(G)$ is given by

$$\phi(G) = 3 \cdot x + y + s.$$

Given an initial connected graph, we take the skeleton T to be a clique tree of the graph. Since the graph is chordal, T can only have as many vertices as the original graph, and thus the potential of a graph is $O(m)$. This potential is used to pay for all the fill introduced by our algorithm. To do this, at each step in the recursion, we ensure that some fixed constant times the difference in potential between the initial graph and the subgraphs in which that step divides the graph is enough to pay for any fill edges that are introduced because of that step (See condition (iii) later in this appendix).

By examining the different kinds of edges in the interval graph, we can determine the following change in the potential function when a new node is inserted in the separator tree.

Lemma 5 *Let I be an interval graph and let P be a rooted skeleton path of I with roots l and r, such that only l is depleted. Let m be a link in the skeleton, distinct from l and r. If m and l are roots in $P(l,m)$, l being depleted in $P(l,m)$, and m and r are roots in $P(m,r)$, m being depleted in $P(m,r)$ (but not in $P(l,m)$) then $\phi(I) - (\phi(P(l,m)) + \phi(P(m,r))) \geq 1 + m'(m'-1) + 2m'(p_m - m')$, where m' is the number of internal vertices that cover m in P.*

Proof. We examine all types of edges, one at a time, and show that the difference in potential comes from making m a root, and the pinning at m of m' vertices that were internal.

Edges that are depleted in I are also depleted in $P(l,m)$. If a depleted edge is present in $P(m,r)$ then its endpoints must cover m, and the edge is also depleted in $P(m,r)$. Therefore, depleted edges in I do not contribute towards potential in either $P(l,m)$ or $P(m,r)$.

Pinned, non-depleted edges in I either occur in one of $P(l,m)$ or $P(m,r)$ or cover m, thus being depleted in $P(m,r)$, again not contributing towards any difference in potential between I and the two subgraphs.

All vertices that are internal to $P(l,m)$ or $P(m,r)$ are also internal to I. Moreover, since they do not cover m they only occur in one of the two subgraphs, as do edges to internal vertices. Therefore, edges to vertices internal to $P(l,m)$ or $P(m,r)$ each contribute 3 units towards the potential of I and to the potential of either $P(l,m)$ or $P(m,r)$, but not both.

Since m is a root in both $P(l,m)$ and $P(m,r)$ but not in I and is the only link that is common to the skeletons of both subgraphs this contributes 1 unit towards the decrease in potential.

The only remaining edges are those from internal edges of I whose endpoints cover m, and are thus pinned in both $P(l,m)$ and $P(m,r)$. They are depleted in $P(m,r)$, but not in $P(l,m)$. Thus they contribute 2 units to the difference in potential. The total number of such edges is at least $m'(m'-1)/2+m'(p_m-m')$, and the total difference in potential is at least $m'(m'-1)+2m'(p_m-m')+1$ ■

We now present an analysis of the performance of our algorithms in terms of fill. The next lemma says that fill is local to the subgraphs defined by the separators in the separator tree.

Lemma 6 *Let I be an interval graph and let P be a rooted skeleton path of I with extremity roots l and r. Then any link m in P distinct from l and r can be inserted into the separator tree, thus dividing I into two subgraphs $P(l,m)$, with roots l and m, and $P(m,r)$, with roots m and r, so that the vertices internal to each of the subgraphs can only have fill to other vertices in that same subgraph.*

Proof. Any path between a vertex that is internal to $P(l,m)$ and another that is internal to $P(m,r)$ must go through some vertex that covers m. Since vertices that cover m are eliminated after all the internal vertices in the elimination order, there cannot be fill between such internal vertices.

The only other possibility is that there exists fill between internal vertices in one subgraph to pinned vertices of the other. If the vertex is pinned at m, then it is in both subgraphs, and there is nothing to prove. Otherwise, any path between the two vertices must still go through some vertex pinned at m. But vertices pinned at m are eliminated after all the vertices which are internal to either subgraph, again contradicting the possibility of fill between the pair of vertices. ■

Let $P(l,r)$ be an interval graph at the beginning of a phase (either homogenize, halving or kill) and let e be a link of its skeleton path P, distinct from its roots l and r. When inserting e into the graph's separator tree we require that:

i. fill between vertices rooted at l and r must have been accounted for in some previous step, if either l or r is depleted, and otherwise must be accounted for in the current step.

ii. fill between vertices pinned at e and those pinned at l (and r) must be accounted for in the current step, if at least one of e or l (respectively r) is depleted.

iii. A constant times the difference in potential between the original graph and the parts must be enough to pay in that step for any fill.

Since condition (iii) refers to the difference in potential, the total amount of fill allowed by (iii) is a constant times the potential of the initial graph and thus $O(m)$. In the next lemmas, we show that the above three invariants are maintained as we perform each of the three procedures described in Section 4.2.

Given an interval graph I, the homogenizing procedure divides the problem of finding an ordering for the vertices of I into a number of subproblems, each of which has the same desirable property, namely except for the extremities, the ply of every link in the skeleton of the subgraphs is at least half of that of the extremity of the skeleton with the largest ply. If this condition is already met, the homogenizing step does not insert any links into the separator tree, and we go into the next step of the algorithm with no depleted links.

Lemma 7 *(Homogenize Step) Let I be an interval graph, with a rooted skeleton path P. Let l and r be roots of P, neither being depleted, and let $p_l \geq p_r$. Then Homogenize(I,P,l,r) finds $k = O(\log p_l)$ links of the skeleton P. If $k > 0$ then the selected links can be inserted into the separator tree in any order, defining subgraphs $P(e_i, e_{i+1})$ rooted at e_i and e_{i+1}, e_i being depleted. Moreover, except for possibly e_i or e_{i+1} the links of the skeleton $P_{e_i,e_{i+1}}$ have ply greater than or equal to half the larger of p_{e_i} and $p_{e_{i+1}}$ and conditions (i), (ii) and (iii) can be satisfied.*

Proof. The whole selection process selects no more than $k \leq \log p_l + \log p_r = O(\log p_l)$ links of the skeleton, aside from l and r. The depletion of the endpoints can be seen as making l depleted, and then applying Lemma 5 repeatedly. Making l depleted reduces the potential of the graph by $p_l(p_l - 1)/2$, while according to Lemma 5 each insertion produces two subgraphs, further reducing the potential by at least 1 unit.

According to Lemma 6, after we insert the first link, say e_i into the separator tree, we obtain two interval graphs $P(l, e_i)$ and $P(e_i, r)$, such that there is no fill between internal vertices in distinct subgraphs. That is, no matter what elimination order we choose for the internal vertices of each subgraph, all the fill to the other subgraph is between pinned vertices.

By repeatedly applying this argument to the subgraphs, and since each link e_i is not depleted in P, and is depleted in $P(e_i, e_{i+1})$, we conclude that the lemma holds, as long as the fill between vertices pinned at the links e_i is smaller than $k + p_l(p_l - 1)/2$. Fill between these vertices includes all the fill between vertices pinned at l and those pinned at r, fill between vertices pinned at e_i and those pinned at e_{i+1}, as well as fill between vertices pinned at non-consecutive links e_i, induced by the ordering among the links e_i.

The total number of vertices pinned at any links e_i distinct from l and r is at most the sum of the plies of the links e_i and that is at most $(p_l + p_r) \leq 2 \cdot p_l$.

There could be fill between vertices pinned at l and those pinned at r, which amounts to at most $p_l \cdot p_r$; let V' be the set of vertices pinned at links e_i other than l and r. There could be fill among the at most $p_l + p_r$ vertices in V', and also from vertices in V' to vertices pinned at l and to vertices pinned at r. These add up to no more than $(p_l + p_r)^2/2$, $(p_l + p_r)p_l$ and $(p_l + p_r)p_r$ respectively.

Since $p_r \leq p_l$, the total number of fill edges is at most $7p_l^2$. For any p_l this is at most a constant times $k + p_l(p_l - 1)/2$. ∎

Specific orderings for the links selected by the homogenizing algorithm can result in smaller constants when computing fill, and in better or worse orderings in terms of the depth of the separator tree.

As long as the skeleton of a subgraph generated by the algorithm described in this lemma has internal vertices, it satisfies all the pre-conditions for Lemma 8 regarding the next step in the algorithm.

Lemma 8 *(Halving Step) Let I be an interval graph, and let P be a rooted skeleton path of I with at least 3 links. Let l and r be its roots, l being possibly depleted. Moreover, let $p_e \geq max(p_l, p_r)/2$, for all links e in P. Then, Halving(I,P,l,r) inserts a link m into the separator tree such that if l is depleted in $P(l,m)$ and m is depleted in $P(m,r)$, conditions (i), (ii) and (iii) are satisfied.*

Proof. As long as P has more than two links the halving procedure will find such a link m. All that needs to be shown is that the fill described in conditions (i) and (ii) can be paid for with a constant times the difference in potential between I and the subgraphs $P(l,m)$ and $P(m,r)$, as prescribed in (iii).

Assume l is not depleted (and neither is any other link of P). Without loss of generality, $p_l \geq p_r$. We can deplete l, thus decreasing the potential of I by $p_l(p_l-1)/2$. The total amount of fill between l and r is no more than p_l^2, and as long as $p_l > 1$ we can pay for that with the decrease in potential resulting from making l depleted. If p_l is 1, then p_r can be at most 1, and there is at most 1 fill edge between vertices pinned at l and vertices pinned at r, which can be paid for when making m a root. The decrease in potential associated with making m a root might also have to be used to pay for edges between vertices that cover m and those pinned at l or r, as follows.

Assume l is depleted, and all fill from vertices pinned at l to those pinned at r has been paid for, by condition (i). When inserting m into the separator tree, we must pay for any fill from vertices pinned at m to those pinned at l or r. Since all fill from l to r has already been accounted for, only fill to vertices that are internal to I and cover m must be considered. Let m' be the number of internal nodes that cover m. Then there are at most $m'(p_l + p_r)$ fill edges induced by the insertion of m in the elimination tree. This fill must be paid for now, to ensure (i), since l is depleted in $P(l,m)$ and m is depleted in $P(m,r)$. Since p_m is at least $max(p_l, p_r)/2$, the amount of fill is at most $4m'.p_m$. Applying Lemma 5 we know that the potential decreases by at least $m'(m'-1) + 2m'(p_m - m') + 1$ when inserting m into the tree, and the amount of fill is at most a constant times larger. ■

For the last step in the algorithm, the depleted root must be the root with largest ply. The following lemma makes sure that is the case, by allowing us to relabel them if needed.

Lemma 9 *Let P be a rooted skeleton of an interval graph. Let l and r be the extremities of P, r being depleted. Then if $p_l \geq p_r$ the depleted status of l and r can be exchanged, without increasing the potential of I.*

Proof. The operation described in the lemma does not affect the number of internal edges. It might decrease the number of pinned, non-depleted edges, thus either reducing or not changing the potential. ■

Coming into the halving procedure, we know that p_e was larger than or equal to $max(p_l, p_r)$. Now, since p_m could be larger than either of those plies, we don't have that condition anymore. However, we can still use the fact that $p_e \geq min\{p_l, p_r\}/2$, where l and r are the extremities of the interval graph being considered.

Lemma 10 *(Kill Step) Let I be an interval graph, with a rooted skeleton path P whose roots are l and r, l being depleted in I. Let $p_l \geq p_r$, and $p_e \geq p_r/2$, for all links e in P. Then Kill(I,P,l,r) finds $k = O(\log p_l)$ links that when inserted into the elimination tree in order, from l to r, satisfy conditions (i) and (iii) while producing subgraphs $K_l(e_i, e_{i+1})$ where e_i and e_{i+1} are non-depleted roots, $0 \leq i \leq k$, and also paying for fill between vertices in $P(e_i, e_{i+1})$ and those in $P(e_i, e_{i+1}) - K_l(e_i, e_{i+1})$ (which includes but is not restricted to the fill described in (ii).)*

Proof. The proof involves examining all possible sources of fill and verifying that they can all be paid for with a constant times the decrease in potential caused by this procedure.

Repeated applications of Lemma 6 show that there is no fill between internal vertices in different subgraphs. The links are inserted in order from l to r into the separator tree, that is, l and r are already in the tree, and then the selected links are inserted in order of increasing distance from l. Therefore, all fill is between vertices pinned in consecutive links or to vertices pinned at r.

Let v be a vertex that has fill to r. Then v is not pinned at r. Moreover, if it is pinned at l, its fill to r has already been paid for, by condition (i). Otherwise, v was a vertex internal to I, and now will occur in at most 2 subgraphs as a pinned vertex (and be removed from any subgraphs whose skeleton it covers entirely). Therefore v will contribute to a decrease in potential. Let e be one of the selected links that v covers. There are least $p_e - 1$ edges to v in I, which are now pinned edges. Each edge might be counted by at most 2 vertices, its endpoints. Since e is now becoming a root it contributes one unit to the decrease in potential, of which v can use up to $1/p_e$. Therefore v can use some constant times $(p_e - 1)/2 + 1/p_e$ to pay for its fill to vertices pinned at r. Since $p_e \geq p_r/2$, a constant will suffice.

The only other source of fill is to those vertices that are removed from the subgraphs. When e_i and e_{i+1} are adjacent, there is no fill to be considered, since all vertices in $P(e_i, e_{i+1})$ are already adjacent. Otherwise, e_i is a sentinel and e_{i+1} is the next milestone (or r). Since neither e_i nor e_{i+1} is depleted in $P(e_i, e_{i+1})$, this step does not need to pay for fill among vertices pinned at e_i and those pinned at e_{i+1} (see (i) and (ii)). However, we do need to pay for fill to vertices that are been removed from the graph, and for that, we will use the potential associated with existing edges to those vertices, which are also being removed.

Two kinds of vertices get removed from the subgraph being defined. If a vertex covers the whole skeleton of the subgraph, it is already adjacent to all its vertices, and there is no fill from within that subgraph to the vertex. Otherwise, the vertex being removed is pinned at l, and covers e_i. Let's consider the vertices in the subgraph $P(e_i, e_{i+1})$ which are not pinned at l. If they are pinned at r, their fill to vertices pinned at l has already been paid for, since l is depleted (from condition (i)). If on the other hand they are internal to I, they might have fill to the vertices that cover e_i and that are being removed from the graph, either because they cover l or because they cover both e_i and e_{i+1}. A number of vertices cover both e_i and l, but at least half as many vertices are pinned at l and cover both e_i and e_{i+1} and are thus adjacent to all vertices of $P(e_i, e_{i+1})$, and in particular to those not pinned at either l or e_i. These edges were internal to I, and thus had 3 units of potential associated with each of them, which can be used to pay for all the fill to edges pinned at both e_i and l. We must make sure that in all other subgraphs $P(e_j, e_{j+1})$, $j \neq i$, each of the edges that were internal and are helping us pay for fill will not be used to pay for fill. But both their endpoints are pinned at e_j and thus they won't be considered. ■

4.3.2 Bounding the amount of work

Unlike the accounting of fill, we must be a bit more careful when computing the work required by a given ordering since the amount of work to eliminate a vertex involves the square of the number of neighbors the vertex has. We need a few lemmas and definitions.

Let $G = (V, E)$ be a graph. c_v is the size of the largest clique of G that contains the vertex $v \in V$. The work involved in eliminating v from G is given by $d + 1 + d^2$, where d is the degree of v in G.

Let V' be the set of internal vertices of G, and let V'' be the set of pinned, non-depleted vertices of G. Moreover, for each vertex v in V'' let p_v be the maximum among the plies of the edges at which v is pinned, and let d_v be the number of depleted vertices that cover that same edge (note that there are at most two edges at which a given vertex might be pinned.) Let $w(x) = x/2 + (x-1)^2$. The *work potential* is defined as $\mathcal{W}(G) = 4 \cdot \sum_{v \in V} (w(c_v) + \sum_{v \in V''} (w(p_v - d_v))$. As the next lemma shows, $\mathcal{W}(G)$ is at most a constant times the minimum amount of work needed to perform Gaussian elimination in G.

Lemma 11 *The total work necessary o perform Gaussian elimination on a graph G is at least $W'(G) = \sum(\frac{c_v}{2} + \frac{(c_v-1)^2}{3})$.*

Proof. The base case, with a single vertex is trivial. Assume the lemma holds for graphs with less than k vertices. Let v be the first vertex to be eliminated in a minimum work elimination of a graph G with k vertices. The lemma holds for $G-\{v\}$. Let d be the degree of v in G. For each of the neighbors of v, either the largest clique it is in has the same size in both $G-\{v\}$ and in G, or it decreases by at most 1 when v is removed, in which case v must be part of that clique. Let w be a neighbor of v in G such that the largest clique that contains w in $G-\{v\}$ has size x, while the largest clique containing w in G has size $x+1$. Then w contributes an extra $1/2+(x^2-(x-1)^2)/3 = 1/2+(2\cdot x-1)/3$ when comparing $W'(G)$ and $W'(G')$. But $x \leq d$ so that the contributions of all neighbors of v to $W'(G)$ add up to at most $d/2+d\cdot(2\cdot d-1)/3$. v's own contribution is at most $(d+1)/2+d^2/3$, so that $W'(G)-W'(G') \leq d+1+d^2$.

By induction, the amount of work necessary to perform Gaussian elimination in G is at least $W'(G')$ plus $d+1+d^2$, the amount of work to eliminate v, and thus at least $W'(G)$. ■

Lemma 12 *Let $P(l,r)$ be an interval graph, with skeleton P, and roots l and r, l being depleted. Let m be an edge of P, distinct from l and r. If l is depleted in $P(l,m)$ and m is depleted in $P(m,r)$ then $\mathcal{W}(P(l,r))-(\mathcal{W}(P(l,m))+\mathcal{W}(P(m,r))) \geq m'\cdot w(p_m)$, where m' is the number of vertices internal to $P(l,r)$ which cover m.*

Proof. Depleted vertices do not contribute to the work potential, and all vertices which are depleted in $P(l,r)$ are depleted in $P(l,m)$. If such a vertex is also in $P(m,r)$, then it must cover m, and is depleted in $P(m,r)$.

Let v be a vertex which is internal to $P(l,m)$. Since v is not adjacent to any vertices which are not in $P(l,m)$, and $P(l,m)$ is an induced subgraph of $P(l,r)$, the largest clique v is in $P(l,r)$ and $P(l,m)$ must have the same size. Vertices which are internal to $P(m,r)$ are analogous. Therefore, vertices internal to either $P(l,m)$ or $P(m,r)$ do not contribute towards the difference in work potential.

The difference in potential is associated with the vertices which are pinned at r and were not depleted in $P(l,r)$ and with those m' vertices that used to be internal, and are now depleted in $P(m,r)$ and pinned (but not depleted) in $P(l,m)$. Let r' be the number of vertices pinned at r and that are not depleted in $P(m,r)$, and let r_m be the number of vertices pinned at both r and m, but not at l. Vertices pinned at m do not contribute to the potential of $P(m,r)$. There are exactly $m'+r_m$ vertices that cover m and are not depleted in $P(l,m)$ thus contributing $(m'+r_m)((m'+r_m)/2+(m'+r_m-1)^2)$ to the potential of $P(l,m)$. Those same m' vertices used to be internal to $P(l,r)$ and were in a clique of size at least p_m, thus adding at least $4\cdot m'\cdot(p_m/2+(p_m-1)^2)$ to the potential of $P(l,r)$. Finally, the r_m+r' vertices which were pinned at r but not depleted in $P(l,r)$ contribute $(r_m+r')((r_m+r')/2+(r_m+r'-1)^2)$ to the work potential of $P(l,r)$, but only contribute $r'(r'/2+(r'-1)^2)$ to the potential of $P(m,r)$ (and have already been accounted for in their contribution towards the potential of $P(l,m)$ as vertices pinned at m and not at l). Thus we have

$$
\begin{aligned}
\mathcal{W}(P(l,r)) - (\mathcal{W}(P(l,m)) + \mathcal{W}(P(m,r)) &\geq \\
4\cdot m'\cdot w(p_m) + (r_m + r')\cdot w(r_m + r') - ((m' + r_m)\cdot w(m' + r_m) + r'\cdot w(r'))
\end{aligned}
$$

The lemma can be proved by showing by subtracting $m' \cdot w(p_m)$ from the quantity above, and showing it is greater than or equal to 0. Since $p_m \geq r_m + m'$ we have:

$$
\begin{aligned}
3\cdot m'\cdot w(p_m) + (r_m + r')\cdot w(r_m + r') - ((m' + r_m)\cdot w(m' + r_m) + r'\cdot w(r')) &\geq \\
3\cdot m'\cdot w(r_m + m') + (r_m + r')\cdot 2(r_m + r') - ((m' + r_m)\cdot w(m' + r_m) + r'\cdot w(r')) &= \\
(3r_m\cdot m'^2 - 3/2\cdot r_m\cdot m') + (2m'^3 + 2m' - 3m'^2) + 3r_m^2\cdot r' + (3r_m\cdot r'^2 - 3r_m\cdot r')
\end{aligned}
$$

But if $m' \geq 0$ we have $3r_m \cdot m'^2 - 3/2 \cdot r_m \cdot m' \geq 0$, as well as $2m'^3 + 2m' - 3m'^2 \geq 0$ (equals 0 for $m' = 0$; equals 1 for $m' = 1$ and is at least m'^2 for $m' \geq 2$). Finally, for $r' \geq 0$ we can also verify that $3r_m \cdot r'^2 - 3r_m \cdot r' \geq 0$, thus concluding the proof. ■

The next lemma allows us to pay for work in parts. All it really says is that as long as we have at most a constant number, in this case 3, of parts that can be paid for separately, then all of them taken together can also be paid for, by increasing a bit the constants involved in the $O()$ notation. Let $\beth'(x) = x + (x-1)^2$.

Lemma 13 *Let G be a graph, and v be a vertex of G. If v has neighbors in at most 3 sets of vertices V_1, V_2 and V_3 by the time v is to be eliminated from G, then the work involved in eliminating v from G is $\beth'(x)$ where $x = |V_1 \cup V_2 \cup V_3|$ which is $O(\beth'(|V_1|) + \beth'(|V_2|) + \beth'(|V_3|))$.*

Proof. Follows from the fact that $|V_1 \cup V_2 \cup V_3|$ is at most 3 times the maximum of $|V_1|$, $|V_2|$ and $|V_3|$. ■

A vertex is said to be *universal* in G if it is adjacent to all vertices of G. Let $W_\pi(G)$ be the work involved in performing Gaussian elimination on the graph G, according to the ordering o. Let $W^*(G)$ be the minimum over all orderings π of $W_\pi(G)$, i.e., the work incurred by a minimum work ordering.

We can now show that the total amount of work required by the ordering provided for our algorithms for both interval and chordal graphs is linear in the minimum amount of work necessary.

Lemma 14 *The total amount of work required to perform Gaussian elimination on an interval graph I according to an ordering generated by our algorithm is $O(W^*(I))$.*

Proof. We shall use the decrease in potential to pay for work involved in eliminating sets of vertices. In each phase, let $P(l,r)$ be the subgraph under consideration. Subgraphs without any internal vertices constitute the base case for the recursion, and are trivial to analyze. Internal vertices of $P(l,r)$ only affect differences in potential between $P(l,r)$ and the new subgraphs created by a phase if they become pinned.

Consider the homogenizing phase: without loss of generality, assume $p_l \geq p_r$. The total number of vertices that are not internal after the end of this phase is at most the sum of the plies of the selected edges, and thus less than $x = 4 \cdot p_l$. Vertices which are internal to any one of the subgraphs defined by this phase are to be eliminated before the vertices that are pinned by the end of this phase. Therefore the work involved in eliminating any one of these pinned vertices is at most $x + (x-1)^2$, not more than a constant times $w(p_l)$. The decrease in potential is in fact at least $w(p_l)$ since l became depleted in this phase and pinning the other vertices does not increase the potential.

Let's consider the halving phase: in the homogenizing phase we have already accounted for the work to eliminate the vertices which are already pinned, so that we only have to account for those that cover the newly selected edge, m, and do not cover either l or r. Let m' be the number of such vertices. They will have edges to the vertices pinned at both l and r.

By Lemma 12, the difference in potential between $P(l,r)$ and the sum of the potentials of $P(l,m)$ and $P(m,r)$ is at least $m' \cdot w(p_m)$. We need to pay for at most $m' \cdot (y + (y-1)^2)$ where $y \leq p_m + p_l + p_r$. Since $p_m \geq max(p_l, p_r)/2$, we can pay for this work with a constant times the decrease in potential.

Next, we must make sure that the work potential does not increase when switching the depleted and non-depleted roots of a skeleton so that we make the root with the largest ply the depleted one. But this follows from the definition of the potential function.

Finally, we consider the kill phase. The work involved in eliminating vertices pinned at l or r has already been accounted for, and we need to account for the work involved in eliminating the vertices that are pinned in this phase.

Assume l is the depleted root. Let v be a vertex that gets pinned at an edge e_i in the kill phase. v will have edges to vertices pinned at the edge e_{i-1} by the time it is to be eliminated (where the e_j's are the edges which were selected in the kill phase, as described in the algorithm). v can also have edges to vertices pinned at r. Lemma 13 tells us we can pay for each part separately. Work associated with vertices pinned at e_{i-1} will be accounted for in the recursive call. v can pay for work associated with r since $p_{e_i} \geq p_r/2$ and the decrease in potential associated with its own pinning is at least $p_{e_i}/2 + (p_{e_i} - 1)^2$.

Finally we have to make sure it is ok to simply remove vertices from the subgraphs $P(e_i, e_{i+1})$ and recurse on $K(e_i, e_{i+1})$. To verify that, we observe that at no other point in the algorithm we change the potential of vertices which are not pinned. If we add all these changes over the various recursive steps of the algorithm, we can actually use $w(c_v)$ as v's contribution towards the potential, where c_v is the size of the largest clique v is in I, instead of considering the size of the largest clique v is in the subgraph being handled.

Consider all vertices that v is adjacent to in the updated graph by the time it is to be eliminated. We have accounted for the work involved in v's elimination due to all vertices but those that were removed in various kill phases. Let y be the number of such vertices. At least half of them were adjacent to v and to themselves. The work due to v's elimination associated with these y vertices is at most a constant times $w(c_v)$, since $c_v \geq y/2 + 1$. Lemma 13 allows us to pay separately for this "part" of the work. ■

5 Chordal graphs

In this section we show how to find an elimination ordering for a chordal graph G by repeatedly applying one of the interval graph algorithms we've already described to branches of G's skeleton. This tree-contraction like approach was also used by Naor, Naor and Schäffer [26] when developing algorithms for chordal graphs.

Consider a chordal graph G, and its skeleton tree T. The algorithm in Section 3 or the algorithm in Section 4 can be used, so that we obtain two chordal graph algorithms that work by finding terminal branches of the skeleton T, and applying the interval graph algorithm to the branches. All pre-existing terminal branches can be processed in parallel, and by re-iterating this process we completely examine the tree in $O(\log n)$ steps.

The actual algorithm that is to be applied to the path $P(l, r)$ is essentially the Kill algorithm. The only differences are that r is not a root, and we do not have any sort of bounds on the plies of the links in the skeleton path. This algorithm will divide $P(l, r)$ in a number of subgraphs, that will then be used as input to our favorite interval graph algorithm.

We add an imaginary link r' to the skeleton of $P(l, r)$, so that r' is now the last link of the skeleton, as opposed to r. Call r' a root. Since $p'_{r'} = 0$, the ply conditions in Lemma 10 are trivially satisfied, and we can apply the Kill algorithm to partition $P(l, r)$ in rooted subgraphs, with no depleted roots. $P(l, r)$ does not include any single-vertex sub-trees that might be present in the chordal graph. These should be eliminated before any of the other vertices of $P(l, r)$, thus not introducing any fill.

Given a graph G, with n vertices and m edges, and whose largest clique has size C, we can show the following results.

- The chordal version of the nested dissection algorithm (Section 3) produces an ordering with $O(C \cdot \log^2 n)$ height and $O(m\sqrt{\log n})$ fill.
- The chordal version of our linear fill algorithm (Section 4) produces an ordering with $O(C \cdot \log^3 n)$ height, $O(m)$ fill, and $O(W^*(G))$ work, where $W^*(G)$ is the minimum amount of work, over all possible orderings for G.

The lemmas presented here allow us to extend the fill analysis to the chordal version of our algorithms.

Lemma 15 *Let G be a chordal graph with a skeleton tree T. Let r be a link to a leaf of T, and l be the other extremity of the terminal branch of T which includes r. Let l be a depleted root in $P(l,r)$ and let G' be the graph obtained from G by removing all vertices of G which are internal to $P(l,r)$. Then the potential $\phi(G) = \phi(G') + \phi(P(l,r)) + 1$.*

Proof. The only vertices that show up in both G' and $P(l,r)$ are those which cover l. Since they are depleted in $P(l,r)$, any edges between them are only counted towards the potential in G'. All other edges of G are internal to either G' or $P(l,r)$, but not both. As for the links, they only show up in either G' or $P(l,r)$, but l is depleted in $P(l,r)$, and therefore doesn't contribute towards $P(l,r)$. ■

Let G, G' and $P(l,r)$ be as described in Lemma 15.

Lemma 16 *If l is the only root in $P(l,r)$, then any ordering for the remaining vertices of $P(l,r)$ does not induce any fill in G to vertices of $G - P(l,r)$.*

Proof. As long as l is the only root in $P(l,r)$, the nodes that cover l must be eliminated after all other vertices of $P(l,r)$ are. Let v be any vertex in $P(l,r)$ that does not cover l, and let w be a vertex of $G - P(l,r)$. Any path from v to w must go through a vertex that covers l, and is thus later in the elimination ordering than w. Therefore, there can't be fill between v and w. ■

Lemma 17 *The chordal graph version of the linear fill algorithm from Section 4 also has linear work.*

Proof. The work potential of a chordal graph is no smaller than the work potential of the graph obtained by removing one of its terminal branches added to the potential of the terminal branch itself, if the edge that used to connect the branch to the skeleton of the graph is a depleted root. This follows since except for the vertices that are depleted, no other vertex appears in both subgraphs. Since the first step of the chordal graph algorithm is identical to a kill operation, in which one of the roots has ply 0, the result follows from the proof of Lemma 14. ■

6 Experimental results

We implemented the chordal graph version of the algorithm presented in Section 5. In this section we present some of the results we have obtained.

The bcsstk matrices used in our experiments come from structural engineering problems, and were obtained from the Harwell-Boeing collection, and from Timothy Davis's "University of Florida Sparse Matrix Collection" (the matrices were provided to Davis by Roger Grimes, at Boeing.) The nasasrb matrix models the structure from the NASA Langley shuttle rocket booster, and the

Matrix	Vertices	Edges
bcsstk30	28924	1007284
bcsstk31	35588	572914
bcsstk32	44609	985046
bcsstk35	30237	709963
bcsstk36	23052	560044
bcsstk37	25503	557737
nasasrb	54870	1311227
G256x256	65536	130560
G64x1024	65536	129984
G16x4096	65536	126960

Table 1: Number of vertices and edges in the original input graphs

G$h \times w$ matrices are $h \times w$ grids. The number of vertices and edges in each of the corresponding graphs can be found in Table 1.

Matrix	Non-zeros				Height			
	AMD	Post AMD	METIS	Post METIS	AMD	Post AMD	METIS	Post METIS
bcsstk30	3853728	4253498	4952030	4880401	2764	2127	1305	1305
bcsstk31	5557200	5911019	5184005	5125629	2285	2334	1428	1424
bcsstk32	4986503	5142573	6817977	6752219	2457	2101	1682	1659
bcsstk35	2732030	2830226	3952484	3901153	1262	1196	1192	1192
bcsstk36	2732511	2766021	3677338	3647743	1540	1435	1348	1348
bcsstk37	2799200	2890112	3679448	3647122	1333	1420	1259	1251
nasasrb	11953582	13816673	11815872	11752835	4829	3617	1874	1874
G256x256	1971395	1995992	2238191	2234147	1617	1581	806	804
G64x1024	1425433	1703195	1750885	1748006	2791	1170	471	464
G16x4096	656963	968911	986596	984855	8464	380	185	182

Table 2: Results obtained by post-processing good existing orderings

We chose to compare our algorithm to a version of the approximate minimum-degree heuristic (AMD) [8] and to METIS [17], which produces nested dissection orderings. Table 2 show the results obtained when using our chordal graph algorithm as a post-processing phase to either AMD (code obtained from ftp://ftp.cise.ufl.edu/pub/umfpack/AMD/) or METIS (code obtained from ftp://ftp.cs.umn.edu/dept/users/kumar/metis/). The number of non-zeros presented in the table includes entries that are in the original, not necessarily chordal, graph, well as any fill entries. Given the final updated graph obtained from a given ordering, we measured the total number of non-zeros above and including the diagonal. The height entries in Table 2 correspond to that of a minimum height ordering that is consistent with the ordering produced, that is, a minimum height ordering whose updated graph is identical to that induced by the corresponding ordering. Given a chordal completion, such a minimum height ordering, and its height can be easily computed [25].

Our results indicate that our algorithm usually produces an ordering that has a small amount of extra fill when compared to the chordal completion it starts with. In some cases, our post-processing actually produces small improvements on the number of non-zeros. Contrary to what we expected, for most graphs, the AMD orderings were very parallel, thus making it harder for us to obtain significant improvements in height. It is interesting to notice that for grids with large aspect ratio the AMD orderings cannot be directly parallelized. In those test cases, our orderings produce a slightly lower number of non-zero entries than the nested dissection orderings, and a small constant higher than the number of non-zeros incurred by the AMD orderings. In these cases, the orderings our algorithm produced are significantly more parallel then the original AMD orderings, and only slightly worse than the nested dissection ones.

The work, that in, number of floating-point operations for each of these orderings can be found in the appendix in Table 3. As we can see, the amount of work for each ordering is highly correlated with the number of non-zeros it induces.

Matrix	Work			
	AMD	Post AMD	METIS	Post METIS
bcsstk30	9.416e+08	1.302e+09	1.488e+09	1.466e+09
bcsstk31	2.898e+09	3.519e+09	1.633e+09	1.622e+09
bcsstk32	9.479e+08	1.036e+09	2.085e+09	2.066e+09
bcsstk35	3.832e+08	4.365e+08	9.785e+08	9.617e+08
bcsstk36	6.201e+08	6.481e+08	1.212e+09	1.206e+09
bcsstk37	5.322e+08	6.021e+08	1.041e+09	1.034e+09
nasasrb	4.771e+09	8.044e+09	4.803e+09	4.787e+09
G256x256	2.607e+08	2.761e+08	3.128e+08	3.126e+08
G64x1024	8.470e+07	1.651e+08	1.325e+08	1.328e+08
G16x4096	8.507e+06	2.653e+07	2.469e+07	2.470e+07

Table 3: Work results obtained by post-processing good existing orderings

7 Concluding remarks

A number of interesting questions remain open:

Can we prove any bounds on the performance of the minimum-degree heuristic on chordal or even interval graphs?

Are there algorithms that produce parallel orderings whose fill is within a constant factor of optimal, for general graphs?

Can we incorporate the idea of sentinels into other heuristics, such as minimum-degree and nested dissection? If so, what bounds can be proved on the fill and work of these algorithms?

References

[1] A. Agrawal, P. Klein, and R. Ravi. Cutting down on fill using nested dissection: provably good elimination orderings. In A. George, J. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 in the IMA Volumes in Mathematics and its Aplications, pages 31–55. Springer–Verlag, 1993.

[2] C. Ashcraft and J. Liu. Robust ordering of sparse matrices using multisection. Technical Report ISSTECH-96-002, Boeing information and support services, 1996.

[3] B. Aspvall. Minimizing elimination tree height can increase fill more than linearly. *Information Processing Letters*, 56:115–120, 1995.

[4] P. Berman and G. Schnitger. On the performance of the minimum degree ordering for gaussian elimination. *SIAM Journal of Matrix Analysis and Applications*, 11(1):83–88, January 1990.

[5] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, March 1995.

[6] P. Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

[7] F. R. K. Chung and D. Mumford. Chordal completions of planar graphs. *Journal of Combinatorial Theory B*, 62(1):96–106, 1994.

[8] T. Davis, P. Amestoy, and I Duff. An approximate minimum degree ordering algorithm. Technical Report TR-94-039, Computer and Information Sciences Dept, University of Florida, December 1994.

[9] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburgh*, 25:71–76, 1961.

[10] K. A. Gallivan et al. *Parallel algorithms for matrix computations.* SIAM, 1990.

[11] F. Gavril. The intersection graph of subtrees of a tree are exactly the chordal graphs. *Journal of Combinatorial Theory B*, 16:47–56, 1974.

[12] J. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal of Numerical Analysis*, 10:345–367, 1973.

[13] J. A. George and J. W. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal of Numerical Analysis*, 15:1053–1069, 1978.

[14] J. R. Gilbert and R. E. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50:377–404, 1987.

[15] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. Technical Report SAND96-0868, Sandia National Labs, March 1996. To appear in SIAM J. Sci. Stat. Comput.

[16] J. Jess and H. Kees. A data structure for parallel L/U decomposition. *IEEE Transactions on Computers*, 31:231–239, 1982.

[17] G. Karypis and V. Kumar. Metis. Unstructured graph partitioning and sparse matrix ordering system. *http://www.www.cs.umn.edu/~karypis/metis/metis.html*, 1995.

[18] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE Computer Society Press, October 1988.

[19] C. Leiserson and J. Lewis. Orderings for parallel sparse symmetric factorization. In G. Rodrigue, editor, *Parallel Processing for Scientific Computing*, pages 27–32. SIAM, Philadelphia, PA, 1987.

[20] J. Lewis, B. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *siamjssc*, 10:1156–1173, 1989.

[21] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16:346–358, 1979.

[22] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[23] J. W. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 12:141–153, 1985.

[24] J. W. Liu. Reordering sparse matrices for parallel elimination. *Parallel Computing*, 11:73–91, 1989.

[25] J. W. Liu and A. Mirzaian. A linear reodering algorithm for parallel pivoting of chordal graphs. *SIAM Journal of Discrete Mathematics*, 2:100–107, 1989.

[26] J. Naor, M. Naor, and A. A. Schäffer. Fast parallel algorithms for chordal graphs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 355–364, 1987.

[27] Victor Pan. Parallel solutions of sparse linear and path systems. In John Reif, editor, *Synthesis of Parallel Algorithms*, pages 621–678. Morgan Kaufmann, 1993.

[28] Victor Pan and John Reif. Efficient parallel solution of linear systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 143–152, Providence,RI, May 1985. ACM.

[29] S. Parter. The use of linear graphs in gaussian elimination. *SIAM review*, 3:364–369, 1961.

[30] A. Pothen. The complexity of optimal elimination trees. Technical Report CS-88-16, Department of Computer Science, The Pennsylvania State University, 1988.

[31] D. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970.

[32] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Algebraic and Discrete Methods*, 2:77–79, 1981.