Parallelizing Elimination Orders with Linear Fill

Claudson Bornstein[1] Bruce Maggs[2] Gary Miller[3] R. Ravi[4]

School of Computer Science[1,2,3] and
Graduate School of Industrial Administration[4]
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents an algorithm for finding parallel elimination orders 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 order 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 order 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 orders 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 order that is actually better, in terms of work and fill, than the original one. We also present an algorithm that produces an order 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 order. For example, suppose that the variables $x_1, \ldots, x_n$ are to be eliminated according to an order 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 orders 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 [32, 34]. 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 order. George [13] first proposed this method for eliminating nodes in a mesh, and later generalized it in a paper with Liu [14] for eliminating the nodes in an arbitrary graph. Bounds on the fill induced by nested dissection orders are known for planar graphs and arbitrary graphs with bounded degree [1, 15, 23].

The *minimum-degree* heuristic repeatedly finds a vertex of minimum degree and eliminates it. This heuristic originated with the work of Markowitz in the

---

[1] E-mail: cfb@cs.cmu.edu

[2] 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 and Sun Microsystems. E-mail: bmm@cs.cmu.edu

[3] Gary Miller is supported in part by NSF Grant CCR-95-05472 and ARPA under Contract N00014-95-1-1246. E-mail: glmiller@cs.cmu.edu

[4] R. Ravi is supported in part by an NSF CAREER Award CCR-96-25297. E-mail: ravi+@andrew.cmu.edu

The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, CMU, or the U.S. Government.

late 50's and has undergone several enhancements in the years since [25]. 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 induced by this heuristic. In fact, there exist graphs for which the fill induced by the minimum-degree order can be very high [4].

Recently, some hybrid algorithms have been experimentally shown to produce orders that compare favorably with those produced by either minimum-degree or nested dissection alone. Hendrickson and Rothberg [16], and independently Liu and Ashcraft [2], proposed algorithms that first find a separator that partitions 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 orders 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 parallel elimination orders for chordal graphs. Chordal graphs are a natural choice, because they are rich in structure and because they are intimately related to Gaussian elimination orders; in fact, chordal graphs are exactly the class of graphs that have zero-fill elimination orders. Moreover, since any elimination order 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 induced by any good ordering algorithm.

Chordal graphs already have zero-fill elimination orders, 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 order, 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 order 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 order has height n, it is often possible to expose some parallelism in a sequential order. The idea is to view a sequential order as a partial order that constrains each vertex to be eliminated before any of its neighbors that appear later in the sequential order. Thus, we can define the height of a sequential order to be the minimum-height parallel order that is consistent with the partial order. Nested dissection is known to produce low-height orders, in particular, within a polylogarithmic factor of the minimum possible [1, 31]. On the other hand, minimum-degree orders 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 order that has simultaneously low height and low fill. Gilbert conjectured the existence of a parallel elimination order that has the minimum possible height among all orders and fill that is only a constant factor more than the number of edges in a minimum-fill order (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 order 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 order with $O(m)$ fill and height close to the minimum possible? In particular, does a nested dissection order accomplish this? We show that the classical nested dissection procedure applied to interval graphs produces an order 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 orders. We exhibit an interval graph algorithm that produces orders 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 orders 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 order 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 induced by our orders 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 orders 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]. In this case, however, the orders 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 order generates an order that has good height and low fill and work. In fact, when compared to a nested dissection order of the original grid, our order 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.

The proofs of the lemmas contained here have been omitted, and can be found in [6].

1.3 Related work: Fill

The problem of finding an elimination order that minimizes the fill for arbitrary graphs is known to be NP-hard [35].

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 [23]. The fill introduced by this variant is $O(n \log n)$ on an n-node graph. Subsequently, Gilbert and Tarjan [15] 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 $O(n \log n)$ [24]. 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 [15, 23] also show a bound of $O(n^{\frac{3}{2}})$ on the work of the orders. It is interesting to note that there are n-node planar graphs (square grids in particular) for which any elimination order introduces fill $\Omega(n \log n)$ [8, 17].

Agrawal, Klein and Ravi [1] gave the first approximation algorithms for finding elimination orders that simultaneously minimize the fill, height and the work, all to within a polylogarithmic factor of optimal when the degree of the input graph is bounded. Their algorithm is essentially the nested dissection algorithm using approximately minimum-size balanced node separators [20] to construct the recursive decomposition. They also analyze the fill and the height of their order 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 order for a given graph with minimum height is NP-hard [33], 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 orders and show how nested dissection can be used for solving the shortest path problem in graphs [31, 30]. 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 [11, 21, 22, 26, 27].

1.5 Outline

The remainder of this paper is organized as follows. In the next section, we introduce some definitions. In Section 3 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 3.1. The second, which has linear fill, is presented in Section 3.2. We then show in Section 4 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 4 can be found in Section 5. We conclude with some remarks in Section 6.

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 without ever really removing any vertices. 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 order 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) in the updated graph iff there exists a path from v to w that only goes through vertices that are eliminated before both v and w.

An order $v_1, v_2, \ldots, v_n$ of the vertices of G is a *perfect elimination order* 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 order. Equivalently, a graph is chordal if every simple cycle with more than three vertices has a chord [10, 34], i.e., no subset of vertices of G induces a subgraph 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 proven 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 [12] and Buneman [7] 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, contains only 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

3.1 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 orders produced.

Given a graph G, an *α-balanced separator* of $G = (V, E)$ is a set of nodes $S \subseteq V$ such that no connected component of $V - S$ has more than $\alpha \cdot |V|$ vertices, for some $\alpha < 1$. 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 corresponds to the set of vertices that cover some link of its skeleton P. We order this tree so that an in-order traversal of the separator tree corresponds to a left-to-right traversal 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 both the left and right subtrees of each node have no more than an α-fraction of the vertices in the subtree rooted at that node.

3.1.1 Analysis

Given a separator tree, we obtain a corresponding elimination order by recursively eliminating the root separator's subtrees in parallel and then eliminating the root separator itself. Even though a nested dissection separator tree has depth $O(\log n)$, the corresponding elimination order potentially can 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 for our nested dissection algorithm.

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

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, so that after this pre-processing step no representative subtree consists of a single node.

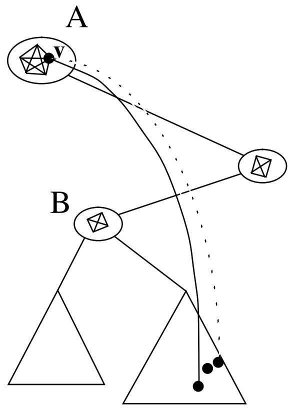

Figure 1: Fill among vertices in a separator tree of an interval graph.

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 order 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 $O(m)$.*

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 $O(\sqrt{k} \cdot \sum_{i=0}^{k} |V_i|^2)$.*

A separator is in at most 4 inner paths, two starting at itself, one starting at its parent, and one starting at some other ancestor. Since 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 order specified by the tree is $O(m \cdot \sqrt{\log n})$.*

It is not hard to find examples of interval graphs on which a nested dissection algorithm that is forced to choose (1/2)-balanced separators produces an ordering with fill $\Omega(m \cdot \sqrt{\log n})$ [6], showing that the bound derived in the previous section is tight.

3.2 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 order for the vertices in the graph. Unlike traditional separator trees, vertices can appear multiple times in the tree, and should be eliminated at the top level separator it appears in.

The algorithm is composed of three steps, 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 steps 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 on which to recurse, each of which has no depleted edges.

The first step, *Homogenize*, finds a sequence of 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 step, 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 steps, thus producing an order with good height. Finally, the algorithm performs the *Kill* step. As mentioned earlier, the purpose of this step is to ensure that the potential function is used correctly to account for the fill. The Kill step accomplishes this by choosing special kinds of separators called sentinels that localize subsequent fill to subgraphs that contain only non-depleted edges (whose potential contribution is as yet unused). In choosing sentinels, we incur an extra $\log n$ factor in the height of our order, since we select up to $\log n$ sentinels per Kill step.

We also analyze a chordal graph version of our interval graph algorithm, which is described in more detail in Section 4. 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 processed.

3.2.1 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 nodes of r. 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. Some subset of the roots may be 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. 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. $P_{l,r}$ denotes the path between and including the two links l and r in T. $P(l,r)$ denotes 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.

3.2.2 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 step 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 interval 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 steps that call each other recursively. Each step inserts some vertices into the separator tree, thus creating subgraphs to which the next step is applied provided there exists at least one vertex internal to the subgraph. Each step operates on an interval graph I, a skeleton path P of I, and P's two extremities, l and r. The first step, Homogenize, divides the interval graph in a number of subgraphs and then applies the Halving step to each one of them. The ply of the links for each subgraph is relatively homogeneous in the sense that internal links have ply at least equal to half of the ply of the roots of that subgraph.

Homogenize(I, P, l, r)

> Select l, and traverse 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 each i between 0 and k for which some vertex in $P(e_i, e_{i+1})$ is not in the separator tree yet.

The Halving step 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$(P(m, r), P_{m,r}, m, r)$), provided that $P(l, m)$ ($P(m, r)$) has some vertex that has not been inserted in the separator tree.

Finally, the Kill step 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 the Homogenize step (as described in Lemma 6), which is applied to each of the resulting subgraphs.

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 node 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.

3.2.3 Fill Analysis

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 $O(m + n)$. 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 into 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 section).

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 4 *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.*

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 5 *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.*

Let $P(l,r)$ be an interval graph at the beginning of a step (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 steps described in Section 3.2.2.

Given an interval graph I, the Homogenize step divides the problem of finding an order 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 Homogenize 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 6 *(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. Sketch: 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 4 repeatedly. Making l depleted reduces the potential of the graph by $p_l(p_l-1)/2$, while according to Lemma 4 each insertion produces two subgraphs, further reducing the potential by at least 1 unit. By considering where fill might be induced, we can verify the amount of fill introduced can be payed for by the decrease in potential. ∎

Specific orders for the links selected by the Homogenize algorithm can result in smaller constants when computing fill, and in better or worse orders 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 7 regarding the next step in the algorithm.

Lemma 7 *(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. Sketch: As long as P has more than two links the Halving step 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). ∎

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 8 *Let P be a rooted skeleton of an interval graph I. 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.*

Coming into the Halving step, 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 9 *(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 separator 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. Sketch: The proof involves examining all possible sources of fill and verifying that they can be paid for with a constant times the decrease in potential caused by this step. ∎

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 steps 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 steps add $O(\log n)$ nodes to the separator tree, and $\log n$ iterations of those three steps 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 order produced has height at most $O(C \cdot \log^2 n)$, where C is the size of the maximum clique of I.

We can obtain similar bounds on the amount of work that is induced by the orders produced by our algorithm. Our results are summarized in the next lemma. The work analysis can be found in [6].

Lemma 10 *Let I be an interval graph with n vertices and m edges. When applied to I, the algorithm presented in Section 3.2.2 produces an order 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 order.*

4 Chordal graphs

In this section we show how to find an elimination order 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 [28] when developing algorithms for chordal graphs.

Using either the algorithm in Section 3.1 or Section 3.2, we can derive an algorithm for chordal graphs. Consider a chordal graph G, and its skeleton tree T. The chordal graph algorithm consists of finding terminal branches of the skeleton T, and applying an interval graph algorithm to the branches. All terminal branches can be processed in parallel. By reiterating this process we order the tree with $O(\log n)$ calls to an interval graph algorithm.

The actual algorithm that is to be applied to the path $P(l, r)$ is essentially the Kill step. 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 an ordering algorithm for interval graphs.

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 9 are trivially satisfied, and we can apply the Kill step to partition $P(l, r)$ in rooted subgraphs, with no depleted roots. $P(l, r)$ does not include any single-vertex subtrees 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.

The lemmas that follow allow us to extend the analysis we have for each of the interval graph algorithms to the chordal case.

Lemma 11 *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 that 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 that are internal to $P(l, r)$. Then the potential $\phi(G) = \phi(G') + \phi(P(l, r)) + 1$.*

Let G, G' and $P(l, r)$ be as described in Lemma 11.

Lemma 12 *If l is the only root in $P(l, r)$, then any order for the remaining vertices of $P(l, r)$ does not induce any fill in G to vertices of $G - P(l, r)$.*

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 3.1) produces an order with $O(C \cdot \log^2 n)$ height and $O(m\sqrt{\log n})$ fill.
- The chordal version of our linear fill algorithm (Section 3.2) produces an order 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 orders for G.

5 Experimental results

We implemented the chordal version of the algorithm presented in Section 4, and performed some experiments.

Matrix	Vertices	Edges
bcsstk30	28924	1007284
bcsstk31	35588	572914
bcsstk32	44609	985046
bcsstk35	30237	709963
bcsstk36	23052	560044
bcsstk37	25503	557737
nasasrb	54870	1311227
sf10	7294	44922
sf5	30169	190377
G256x256	65536	130560
G64x1024	65536	129984
G16x4096	65536	126960

Table 1: Number of vertices and edges of the input graphs.

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

Matrix	Non-zeros	Non-zeros / AMD Non-zeros					Height	Height / AMD Height				
	AMD	Post AMD	METIS	Post METIS	Hybrid	Post hybrid	AMD	Post AMD	METIS	Post METIS	Hybrid	Post Hybrid
bcsstk30	3853728	1.1037	1.2850	1.2664	1.0098	1.0307	2764	0.77	0.47	0.47	0.65	0.60
bcsstk31	5557200	1.0637	0.9328	0.9223	0.7527	0.7638	2285	1.02	0.62	0.62	0.72	0.72
bcsstk32	4986503	1.0313	1.3673	1.3541	1.0162	1.0339	2457	0.86	0.68	0.68	0.72	0.68
bcsstk35	2732030	1.0359	1.4467	1.4279	1.0160	1.0268	1262	0.95	0.94	0.94	0.86	0.85
bcsstk36	2732511	1.0123	1.3458	1.3349	0.9350	0.9527	1540	0.93	0.88	0.88	0.81	0.85
bcsstk37	2799200	1.0325	1.3145	1.3029	0.9620	0.9785	1333	1.07	0.94	0.94	0.87	0.89
nasasrb	11953582	1.1559	0.9885	0.9832	0.8169	0.8529	4829	0.75	0.39	0.39	0.70	0.58
sf10	675708	1.0223	0.8766	0.8748	0.7858	0.7916	793	1.00	0.71	0.69	0.89	0.87
sf5	5244006	1.0203	0.8045	0.8039	0.7385	0.8001	2341	1.00	0.61	0.60	0.78	0.81
G256x256	1971395	1.0125	1.1353	1.1333	0.8495	0.8549	1617	0.98	0.50	0.50	0.62	0.61
G64x1024	1425433	1.1949	1.2283	1.2263	0.9476	0.9641	2791	0.42	0.17	0.17	0.35	0.28
G16x4096	656963	1.4748	1.5018	1.4991	1.3240	1.3624	8464	0.04	0.02	0.02	0.09	0.04

Table 2: Results obtained by post-processing good existing orders relative to an AMD order.

Matrix	AMD Work	Work / AMD Work				
	AMD	Post AMD	METIS	Post METIS	Hybrid	Post Hybrid
bcsstk30	9.416e+08	1.3828	1.5803	1.5569	0.9944	1.0631
bcsstk31	2.898e+09	1.2143	0.5635	0.5597	0.3999	0.4179
bcsstk32	9.479e+08	1.0929	2.1996	2.1796	1.0116	1.0708
bcsstk35	3.832e+08	1.1391	2.5535	2.5097	1.0287	1.0558
bcsstk36	6.201e+08	1.0452	1.9545	1.9448	0.7792	0.8429
bcsstk37	5.322e+08	1.1313	1.9560	1.9429	0.8968	0.9605
nasasrb	4.771e+09	1.6860	1.0067	1.0034	0.5911	0.6866
sf10	1.366e+08	1.0681	0.6400	0.6395	0.5178	0.5279
sf5	2.781e+09	1.0550	0.5358	0.5358	0.4585	0.6228
G256x256	2.607e+08	1.0591	1.1998	1.1991	0.7280	0.7484
G64x1024	8.470e+07	1.9492	1.5643	1.5679	0.9843	1.0581
G16x4096	8.507e+06	3.1186	2.9023	2.9035	2.4168	2.6284

Table 3: Work results obtained by post-processing good existing orders relative to an AMD order.

Roger Grimes, at Boeing.) The nasasrb matrix models the structure of the NASA Langley shuttle rocket booster, while the sf matrices are used in the simulation of an earthquake in the San Fernando Valley [29]. The G matrices are $h \times w$ grids. The number of vertices and edges in each graph can be found in Table 1.

We applied our algorithm as a post processing step to the orders produced by a version of the approximate minimum-degree heuristic (AMD) [2] [9], to the nested dissection orders produced by METIS [3] [19] and to the orders produced by a hybrid of min-degree and nested dissection obtained from Rothberg [16].

Table 2 shows the results we obtained as a function of the numbers obtained for an AMD order. The number of non-zeros includes entries that are in the original graph as well as any fill entries, above and including the diagonal. The height entries correspond to the height of a minimum height order whose updated graph is identical to the chordal completion being considered. Given a chordal completion, such a minimum height order and its height can be easily computed (see [18, 27]).

Table 3 shows the amount of work, that is, floating-point operations, involved in performing Gaussian elimination according to each of the orders.

Our results indicate that our algorithm usually produces an order 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 orders 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 orders cannot be directly parallelized. In those test cases, our orders induce a slightly lower number of non-zero entries than the nested dissection orders, and a small constant factor higher than the number of non-zeros induced by the AMD orders. In these cases, the orders our algorithm produced are significantly more parallel then the original AMD orders, and only slightly worse than the nested dissection orders. The hybrid algorithm produces orders that are usually good in terms of both fill and height.

6 Concluding remarks

A number of interesting questions remain open:

Can we prove any bounds on the performance of

[2] code from ftp://ftp.cise.ufl.edu/pub/umfpack/AMD/

[3] code from ftp://ftp.cs.umn.edu/dept/users/kumar/metis/

the minimum-degree heuristic on chordal or even interval graphs?

Are there algorithms that obtain parallel orders with fill 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 fill and work bounds can be proved?

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 of the IMA Volumes in Mathematics and its Applications, pages 31–55. Springer–Verlag, New York, NY, 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] C. Bornstein, B. Maggs, G. Miller, and R. Ravi. Parallel Gaussian elimination with linear work and fill. Technical Report CMU–CS–97–133, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1997.

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

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

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

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

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

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

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

[14] 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.

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

[16] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. Technical Report SAND96-0868, Sandia National Labs, Albuquerque, NM, March 1996. To appear in the SIAM Journal on Scientific and Statistical Computing.

[17] A. Hoffmann, M. Martin, and D. Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM Journal of Numerical Analysis*, 10:364–369, 1973.

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

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

[20] 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, October 1988.

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

[22] J. Lewis, B. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM Journal on Scientific and Statistical Computing*, 10:1156–1173, 1989.

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

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

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

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

[27] 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.

[28] 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.

[29] D. O'Hallaron and J. Shewchuk. Properties of a family of parallel finite element simulations. Technical Report CMU-CS-96-141, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996.

[30] V. Pan. Parallel solutions of sparse linear and path systems. In John Reif, editor, *Synthesis of Parallel Algorithms*, pages 621–678. Morgan Kaufmann, San Mateo, CA, 1993.

[31] V. Pan and J. 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.

[32] S. Parter. The use of linear graphs in Gaussian elimination. *SIAM Review*, 3:364–369, 1961.

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

[34] D. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

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