AN APPROXIMATE MINIMUM DEGREE ORDERING ALGORITHM

TIMOTHY A. DAVIS*, PATRICK AMESTOY†, AND IAIN S. DUFF‡

Computer and Information Sciences Dept., University of Florida,
Technical Report TR-94-039, December, 1994 (revised July 1995).

Abstract. An Approximate Minimum Degree ordering algorithm (AMD) for preordering a symmetric sparse matrix prior to numerical factorization is presented. We use techniques based on the quotient graph for matrix factorization that allow us to obtain computationally cheap bounds for the minimum degree. We show that these bounds are often equal to the actual degree. The resulting algorithm is typically much faster than previous minimum degree ordering algorithms, and produces results that are comparable in quality with the best orderings from other minimum degree algorithms.

Keywords: approximate minimum degree ordering algorithm, quotient graph, sparse matrices, graph algorithms, ordering algorithms

AMS classifications: 65F50, 65F05.

1. Introduction. When solving large sparse symmetric linear systems of the form $\mathbf{Ax} = \mathbf{b}$, it is common to precede the numerical factorization by a symmetric reordering. This reordering is chosen so that pivoting down the diagonal in order on the resulting permuted matrix $\mathbf{PAP}^T = \mathbf{LL}^T$ produces much less fill-in and work than computing the factors of $\mathbf{A}$ by pivoting down the diagonal in the original order. This reordering is computed using only information on the matrix structure without taking account of numerical values and so may not be stable for general matrices. However, if the matrix $\mathbf{A}$ is positive-definite [21], a Cholesky factorization can safely be used. This technique of preceding the numerical factorization with a symbolic analysis can also be extended to unsymmetric systems although the numerical factorization phase must allow for subsequent numerical pivoting [1, 2, 16]. The goal of the preordering is to find a permutation matrix $\mathbf{P}$ so that the subsequent factorization has the least fill-in. Unfortunately, this problem is NP-complete [31], so heuristics are used.

The minimum degree ordering algorithm is one of the most widely used heuristics, since it produces factors with relatively low fill-in on a wide range of matrices. Because of this, the algorithm has received much attention over the past three decades. The algorithm is a symmetric analogue of Markowitz' method [26] and was first proposed by Tinney and Walker [30] as algorithm S2. Rose [27, 28] developed a graph theoretical model of Tinney and Walker's algorithm and renamed it the minimum degree algorithm, since it performs its pivot selection by selecting from a graph a node of minimum degree. Later implementations have dramatically improved the time and memory requirements of Tinney and Walker's method, while maintaining the basic idea of selecting a node or set of nodes of minimum degree. These improvements have reduced the memory complexity so that the algorithm can operate within the storage of the original matrix, and have reduced the amount of work needed to keep track of the degrees of nodes in the graph (which is the most computationally intensive part

* Computer and Information Sciences Department University of Florida, Gainesville, Florida, USA. phone: (904) 392-1481, email: davis@cis.ufl.edu. Support for this project was provided by the National Science Foundation (ASC-9111263 and DMS-9223088). Portions of this work were supported by a post-doctoral grant from CERFACS.

† ENSEEIHT-IRIT, Toulouse, France. email: amestoy@enseeiht.fr.

‡ Rutherford Appleton Laboratory, Chilton, Didcot, Oxon. 0X11 0QX England, and European Center for Research and Advanced Training in Scientific Computation (CERFACS), Toulouse, France. email: isd@letterbox.rl.ac.uk.

of the algorithm). This work includes that of Duff and Reid [10, 13, 14, 15]; George and McIntyre [23]; Eisenstat, Gursky, Schultz, and Sherman [17, 18]; George and Liu [19, 20, 21, 22]; and Liu [25]. More recently, several researchers have relaxed this heuristic by computing upper bounds on the degrees, rather than the exact degrees, and selecting a node of minimum upper bound on the degree. This work includes that of Gilbert, Moler, and Schreiber [24], and Davis and Duff [7, 8]. Davis and Duff use degree bounds in the unsymmetric-pattern multifrontal method (UMFPACK), an unsymmetric Markowitz-style algorithm. In this paper, we describe an approximate minimum degree ordering algorithm based on the symmetric analogue of the degree bounds used in UMFPACK.

Section 2 presents the original minimum degree algorithm of Tinney and Walker in the context of the graph model of Rose. Section 3 discusses the quotient graph (or element graph) model and the use of that model to reduce the time taken by the algorithm. In this context, we present our notation for the quotient graph, and present a small example matrix and its graphs. We then use the notation to describe our approximate degree bounds in Section 4. The Approximate Minimum Degree (AMD) algorithm and its time complexity is presented in Section 5. In Section 6, we first analyse the performance and accuracy of our approximate degree bounds on a set of test matrices from a wide range of disciplines. The AMD algorithm is then compared with other established codes that compute minimum degree orderings.

2. Elimination graphs. The nonzero pattern of a symmetric n-by-n matrix, $\mathbf{A}$, can be represented by a graph $G^0 = (V^0, E^0)$, with nodes $V^0 = \{1, ..., n\}$ and edges E^0. An edge (i, j) is in E^0 if and only if $a_{ij} \neq 0$. Since $\mathbf{A}$ is symmetric, G^0 is undirected.

The elimination graph, $G^k = (V^k, E^k)$, describes the nonzero pattern of the submatrix still to be factorized after the first k pivots have been chosen. It is undirected, since the matrix remains symmetric as it is factorized. At step k, the graph G^k depends on G^{k-1} and the selection of the kth pivot. To find G^k, the kth pivot node p is selected from V^{k-1}. Edges are added to E^{k-1} to make the nodes adjacent to p in G^{k-1} a *clique* (a fully connected subgraph). This addition of edges (fill-in) means that we cannot know the storage requirements in advance. The edges added correspond to fill-in caused by the kth step of factorization. A fill-in is a nonzero entry $\mathbf{L}_{ij}$, where $(\mathbf{PAP}^T)_{ij}$ is zero. The pivot node p and its incident edges are then removed from the graph G^{k-1} to yield the graph G^k. Let $\mathrm{Adj}_{G^k}(i)$ denote the set of nodes adjacent to i in the graph G^k. Throughout this paper, we will use the superscript k to denote a graph, set, or other structure obtained after the first k pivots have been chosen. For simplicity, we will drop the superscript when the context is clear.

The minimum degree algorithm selects node p as the kth pivot such that the degree of p, $t_p \equiv |\mathrm{Adj}_{G^{k-1}}(p)|$, is minimized (where $|...|$ denotes the size of a set or the number of nonzeros in a matrix). The minimum degree algorithm is a non-optimal greedy heuristic for reducing the number of new edges (fill-ins) introduced during the factorization. We have already noted that the optimal solution is NP-complete [31]. By minimizing the degree, the algorithm minimizes the upper bound on the fill-in caused by the kth pivot. Selecting p as pivot creates at most $(t_p^2 - t_p)/2$ new edges in G.

3. Quotient graphs. In contrast to the elimination graph, the quotient graph models the factorization of $\mathbf{A}$ using an amount of storage that never exceeds the storage for the original graph, G^0 [21]. The quotient graph is also referred to as the

generalized element model [13, 14, 15, 29]. An important component of a quotient graph is a clique. It is a particularly economic structure since a clique is represented by a list of its members rather than by a list of all the edges in the clique. Following the generalized element model, we refer to nodes removed from the elimination graph as *elements* (George and Liu refer to them as eliminated nodes). We use the term *variable* to refer to uneliminated nodes.

The quotient graph, $\mathcal{G}^k = (V^k, \overline{V}^k, E^k, \overline{E}^k)$, implicitly represents the elimination graph G^k, where $\mathcal{G}^0 = G^0$. For clarity, we drop the superscript k in the following. The nodes in $\mathcal{G}$ consist of variables (the set V), and elements (the set $\overline{V}$). The edges are divided into two sets: edges between variables $E \subseteq V \times V$, and between variables and elements $\overline{E} \subseteq V \times \overline{V}$. There are no edges between elements since they are removed by element absorption. The sets $\overline{V}^0$ and $\overline{E}^0$ are empty.

We use the following set notation ($\mathcal{A}$, $\mathcal{E}$, and $\mathcal{L}$) to describe the quotient graph model and our approximate degree bounds. Let $\mathcal{A}_i$ be the set of variables adjacent to variable i in $\mathcal{G}$, and let $\mathcal{E}_i$ be the set of elements adjacent to variable i in $\mathcal{G}$ (we refer to $\mathcal{E}_i$ as element list i). That is, if i is a variable in V, then

$$\mathcal{A}_i \equiv \{j : (i,j) \in E\} \subseteq V,$$

$$\mathcal{E}_i \equiv \{e : (i,e) \in \overline{E}\} \subseteq \overline{V},$$

and

$$\mathrm{Adj}_{\mathcal{G}}(i) \equiv \mathcal{A}_i \cup \mathcal{E}_i \subseteq V \cup \overline{V}.$$

The set $\mathcal{A}_i$ refers to a subset of the nonzero entries in row i of the original matrix $\mathbf{A}$ (thus the notation $\mathcal{A}$). That is, $\mathcal{A}_i^0 \equiv \{j : a_{ij} \neq 0\}$, and $\mathcal{A}_i^k \subseteq \mathcal{A}_i^{k-1}$, for $1 \leq k \leq n$. Let $\mathcal{L}_e$ denote the set of variables adjacent to element e in $\mathcal{G}$. That is, if e is an element in $\overline{V}$, then we define

$$\mathcal{L}_e \equiv \mathrm{Adj}_{\mathcal{G}}(e) = \{i : (i,e) \in \overline{E}\} \subseteq V.$$

The edges E and $\overline{E}$ in the quotient graph are represented explicitly as the sets $\mathcal{A}_i$ and $\mathcal{E}_i$ for each variable in $\mathcal{G}$, and the sets $\mathcal{L}_e$ for each element in $\mathcal{G}$. We will use $\mathcal{A}$, $\mathcal{E}$, and $\mathcal{L}$ to denote three sets containing all $\mathcal{A}_i$, $\mathcal{E}_i$, and $\mathcal{L}_e$, respectively, for all variables i and all elements e. George and Liu [21] show that the quotient graph takes no more storage than the original graph ($|\mathcal{A}^k| + |\mathcal{E}^k| + |\mathcal{L}^k| \leq |\mathcal{A}^0|$ for all k).

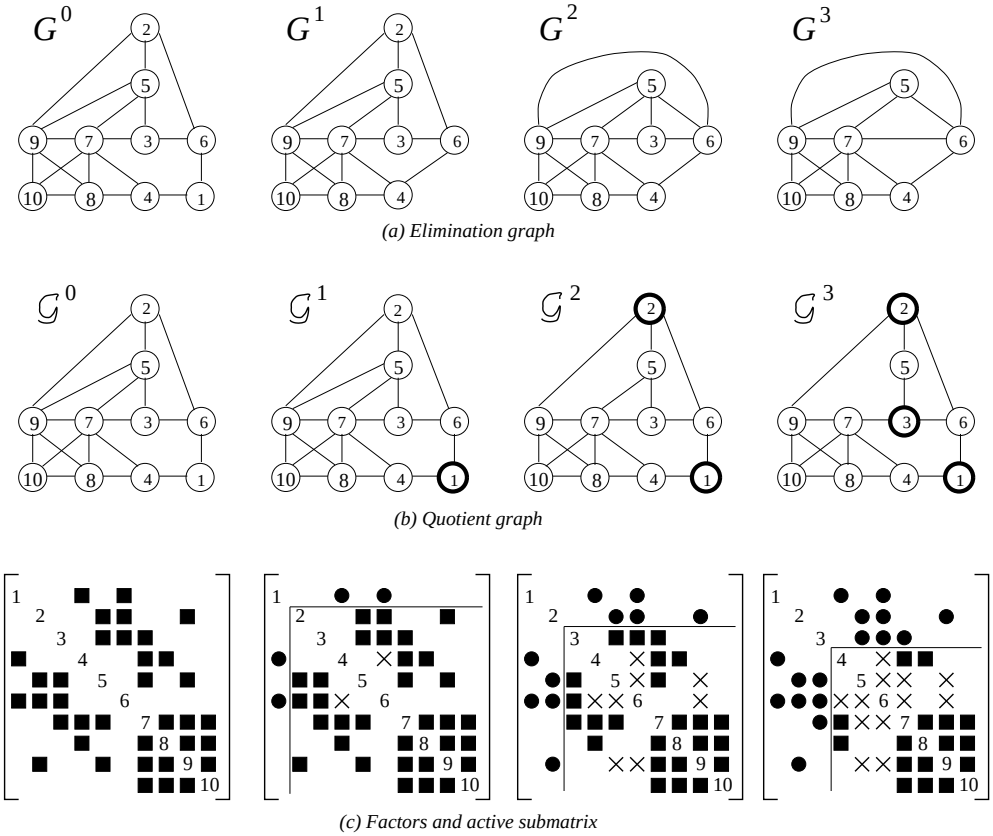
The quotient graph $\mathcal{G}$ and the elimination graph G are closely related. If i is a variable in G, it is also a variable in $\mathcal{G}$, and

$$\mathrm{Adj}_G(i) = \left(\mathcal{A}_i \cup \bigcup_{e \in \mathcal{E}_i} \mathcal{L}_e \right) \setminus \{i\}, \tag{3.1}$$

where the "\" is the standard set subtraction operator.

When variable p is selected as the kth pivot, element p is formed (variable p is removed from V and added to $\overline{V}$). The set $\mathcal{L}_p = \mathrm{Adj}_G(p)$ is found using Equation (3.1). The set $\mathcal{L}_p$ represents a permuted nonzero pattern of the kth column of $\mathbf{L}$ (thus the notation $\mathcal{L}$). If $i \in \mathcal{L}_p$, where p is the kth pivot, and variable i will become the mth pivot (for some $m > k$), then the entry $\mathbf{L}_{mk}$ will be nonzero. Equation (3.1) implies that $\mathcal{L}_e \setminus \{p\} \subseteq \mathcal{L}_p$ for all elements e adjacent to variable p. This means that all variables adjacent to an element $e \in \mathcal{E}_p$ are adjacent to the element p and these elements

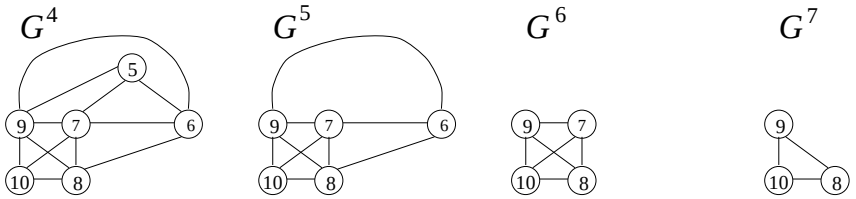
FIG. 3.1. *Elimination graph, quotient graph, and matrix for first three steps.*



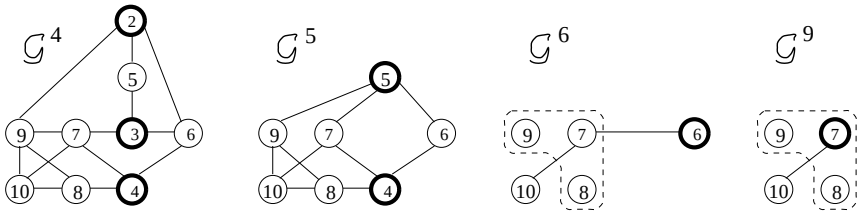
$e \in \mathcal{E}_p$ are no longer needed. They are *absorbed* into the new element p and deleted [15], and reference to them is replaced by reference to the new element p. The new element p is added to the element lists, $\mathcal{E}_i$, for all variables i adjacent to element p. Absorbed elements, $e \in \mathcal{E}_p$, are removed from all element lists. The sets $\mathcal{A}_p$ and $\mathcal{E}_p$, and $\mathcal{L}_e$ for all e in $\mathcal{E}_p$, are deleted. Finally, any entry j in $\mathcal{A}_i$, where both i and j are in $\mathcal{L}_p$, is redundant and is deleted. The set $\mathcal{A}_i$ is thus disjoint with any set $\mathcal{L}_e$ for $e \in \mathcal{E}_i$. In other words, $\mathcal{A}_i^k$ is the pattern of those entries in row i of $\mathbf{A}$ that are not modified by steps 1 through k of the Cholesky factorization of $\mathbf{PAP}^T$. The net result is that the new graph $\mathcal{G}$ takes the same, or less, storage than before the kth pivot was selected.

3.1. Quotient graph example. We illustrate the sequence of elimination graphs and quotient graphs of a 10-by-10 sparse matrix in Figures 3.1 and 3.2. The example is ordered so that a minimum degree algorithm recommends pivoting down the diagonal in the natural order (that is, the permutation matrix is the identity). In Figures 3.1 and 3.2, variables and elements are shown as thin-lined and heavy-lined circles, respectively. In the matrices in these figures, diagonal entries are numbered, unmodified original nonzero entries (entries in $\mathcal{A}$) are shown as a solid squares. The solid squares in row i form the set $\mathcal{A}_i$. The variables in current unabsorbed elements (sets $\mathcal{L}_e$) are indicated by solid circles in the columns of $\mathbf{L}$ corresponding to the unabsorbed

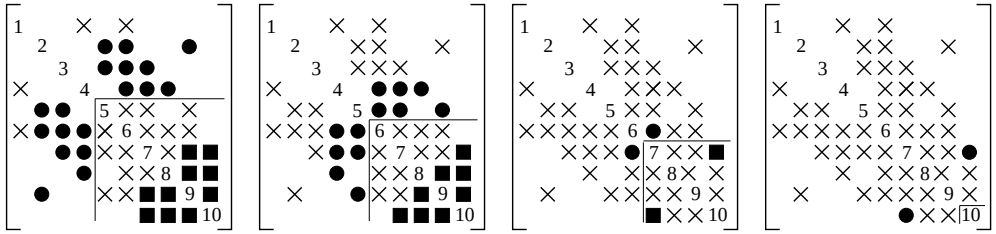
FIG. 3.2. *Elimination graph, quotient graph, and matrix for steps 4 to 7.*



(a) Elimination graph



(b) Quotient graph



(c) Factors and active submatrix

elements. The solid circles in row i form the set $\mathcal{E}_i$. Entries that do not correspond to edges in the quotient graph are shown as an $\times$. Figure 3.1 shows the elimination graph, quotient graph, and the matrix prior to elimination (in the left column) and after the first three steps (from left to right). Figure 3.2 continues the example for the next four steps.

Consider the transformation of the graph $\mathcal{G}^2$ to the graph $\mathcal{G}^3$. Variable 3 is selected as pivot. We have $\mathcal{L}_3 = \mathcal{A}_3 = \{5, 6, 7\}$ (a simple case of Equation (3.1)). The new element 3 represents the pairwise adjacency of variables 5, 6, and 7. The explicit edge (5,7) is now redundant, and is deleted from $\mathcal{A}_5$ and $\mathcal{A}_7$.

Also consider the transformation of the graph $\mathcal{G}^4$ to the graph $\mathcal{G}^5$. Variable 5 is selected as pivot. The set $\mathcal{A}_5$ is empty and $\mathcal{E}_5 = \{2, 3\}$. Following Equation (3.1),

$$\begin{aligned} \mathcal{L}_5 &= (\mathcal{A}_5 \cup \mathcal{L}_2 \cup \mathcal{L}_3) \setminus \{5\} \\ &= (\emptyset \cup \{5, 6, 9\} \cup \{5, 6, 7\}) \setminus \{5\} \\ &= \{6, 7, 9\}, \end{aligned}$$

which is the pattern of column 5 of $\mathbf{L}$ (excluding the diagonal). Since the new element 5 implies that variables 6, 7, and, 9 are pairwise adjacent, elements 2 and 3 do not add any information to the graph. They are removed, having been "absorbed" into element 5. Additionally, the edge (7, 9) is redundant, and is removed from $\mathcal{A}_7$ and

$\mathcal{A}_9$. In $\mathcal{G}^4$, we have

$$\begin{array}{ll} \mathcal{A}_6 = \emptyset & \mathcal{E}_6 = \{2,3,4\} \\ \mathcal{A}_7 = \{9,10\} & \mathcal{E}_7 = \{3,4\} \\ \mathcal{A}_9 = \{7,8,10\} & \mathcal{E}_9 = \{2\} \end{array} .$$

After these transformations, we have in $\mathcal{G}^5$,

$$\begin{array}{ll} \mathcal{A}_6 = \emptyset & \mathcal{E}_6 = \{4,5\} \\ \mathcal{A}_7 = \{10\} & \mathcal{E}_7 = \{4,5\} \\ \mathcal{A}_9 = \{8,10\} & \mathcal{E}_9 = \{5\} \end{array} ,$$

and the new element in $\mathcal{G}^5$,

$$\mathcal{L}_5 = \{6,7,9\}.$$

3.2. Indistinguishable variables and external degree. Two variables i and j are *indistinguishable* in G if $\mathrm{Adj}_G(i) \cup \{i\} = \mathrm{Adj}_G(j) \cup \{j\}$. They will have the same degree until one is selected as pivot. If i is selected, then j can be selected next without causing any additional fill-in. Selecting i and j together is called *mass elimination* [23]. Variables i and j are replaced in $\mathcal{G}$ by a *supervariable* containing both i and j, labeled by its *principal* variable (i, say) [13, 14, 15]. Variables that are not supervariables are called *simple* variables. In practice, new supervariables are constructed at step k only if both i and j are in $\mathcal{L}_p$ (where p is the pivot selected at step k). In addition, rather than checking the graph G for indistinguishability, we use the quotient graph $\mathcal{G}$ so that two variables i and j are found to be indistinguishable if $\mathrm{Adj}_{\mathcal{G}}(i) \cup \{i\} = \mathrm{Adj}_{\mathcal{G}}(j) \cup \{j\}$. This comparison is faster than determining if two variables are indistinguishable in G, but may miss some identifications because, although indistinguishability in $\mathcal{G}$ implies indistinguishability in G, the reverse is not true.

We denote the set of simple variables in the supervariable with principal variable i as $\mathbf{i}$, and define $\mathbf{i} = \{i\}$ if i is a simple variable. When p is selected as pivot at the kth step, all variables in $\mathbf{p}$ are eliminated. The use of supervariables greatly reduces the number of degree computations performed, which is the most costly part of the algorithm. Non-principal variables and their incident edges are removed from the quotient graph data structure when they are detected. The set notation $\mathcal{A}$ and $\mathcal{L}$ refers either to a set of supervariables or to the variables represented by the supervariables, depending on the context. In degree computations and when used in representing elimination graphs, the sets refer to variables; otherwise they refer to supervariables.

In Figure 3.2, detected supervariables are circled by dashed lines. Non-principal variables are left inside the dashed supervariables. These are, however, removed from the quotient graph. The last quotient graph in Figure 3.2 represents the selection of pivots 7, 8, and 9, and thus the right column of the figure depicts G^7, $\mathcal{G}^9$, and the matrix after the ninth pivot step.

The *external* degree $d_i \equiv t_i - |\mathbf{i}| + 1$ of a principal variable i is

$$d_i = |\mathrm{Adj}_G(i) \setminus \mathbf{i}| = |\mathcal{A}_i \setminus \mathbf{i}| + \left| \left(\bigcup_{e \in \mathcal{E}_i} \mathcal{L}_e \right) \setminus \mathbf{i} \right|, \tag{3.2}$$

since the set $\mathcal{A}_i$ is disjoint from any set $\mathcal{L}_e$ for $e \in \mathcal{E}_i$. At most $(d_i^2 - d_i)/2$ fill-ins occur if all variables in $\mathbf{i}$ are selected as pivots. We refer to t_i as the true degree of variable i. Selecting the pivot with minimum external degree tends to produce a better ordering than selecting the pivot with minimum true degree [25] (also see Section 6.2).

Algorithm 1 (Minimum degree algorithm, based on quotient graph)

$V = \{1 \ldots n\}$
$\overline{V} = \emptyset$
for $i = 1$ **to** n **do**
  $\mathcal{A}_i = \{j : a_{ij} \neq 0 \text{ and } i \neq j\}$
  $\mathcal{E}_i = \emptyset$
  $d_i = |\mathcal{A}_i|$
  $\mathbf{i} = \{i\}$
end for
$k = 1$
while $k \leq n$ **do**
  mass elimination:
  select variable $p \in V$ that minimizes d_p
  $\mathcal{L}_p = \left(\mathcal{A}_p \cup \bigcup_{e \in \mathcal{E}_p} \mathcal{L}_e\right) \setminus \mathbf{p}$
  for each $\mathbf{i} \in \mathcal{L}_p$ **do**
    remove redundant entries:
    $\mathcal{A}_i = (\mathcal{A}_i \setminus \mathcal{L}_p) \setminus \mathbf{p}$
    element absorption:
    $\mathcal{E}_i = (\mathcal{E}_i \setminus \mathcal{E}_p) \cup \{p\}$
    compute external degree:
    $d_i = |\mathcal{A}_i \setminus \mathbf{i}| + \left|\left(\bigcup_{e \in \mathcal{E}_i} \mathcal{L}_e\right) \setminus \mathbf{i}\right|$
  end for
  supervariable detection, pairs found via hash function:
  for each pair $\mathbf{i}$ and $\mathbf{j} \in \mathcal{L}_p$ **do**
    if i and **j** are indistinguishable **then**
      remove the supervariable **j**:
      $\mathbf{i} = \mathbf{i} \cup \mathbf{j}$
      $d_i = d_i - |\mathbf{j}|$
      $V = V \setminus \{j\}$
      $\mathcal{A}_j = \emptyset$
      $\mathcal{E}_j = \emptyset$
    end if
  end for
  convert variable p *to element* p:
  $\overline{V} = (\overline{V} \cup \{p\}) \setminus \mathcal{E}_p$
  $V = V \setminus \{p\}$
  $\mathcal{A}_p = \emptyset$
  $\mathcal{E}_p = \emptyset$
  $k = k + |\mathbf{p}|$
end while

3.3. Quotient-graph-based minimum degree algorithm. A minimum degree algorithm based on the quotient graph is shown in Algorithm 1. It includes element absorption, mass elimination, supervariables, and external degrees. Supervariable detection is simplified by computing a hash function on each variable, so that not all pairs of variables need be compared [3]. Algorithm 1 does not include two

important features of Liu's *Multiple Minimum Degree* algorithm (MMD): incomplete update [17, 18] and multiple elimination [25]. With multiple elimination, an independent set of pivots with minimum degree is selected before any degrees are updated. If a variable is adjacent to two or more pivot elements, its degree is computed only once. A variable j is *outmatched* if $\mathrm{Adj}_G(i) \subseteq \mathrm{Adj}_G(j)$. With incomplete degree update, the degree update of the outmatched variable j is avoided until variable i is selected as pivot. These two features further reduce the amount of work needed for the degree computation in MMD. We will discuss their relationship to the AMD algorithm in the next section.

The time taken to compute d_i using Equation (3.2) by a quotient-graph-based minimum degree algorithm is

$$\Theta(|\mathcal{A}_i| + \sum_{e \in \mathcal{E}_i} |\mathcal{L}_e|), \tag{3.3}$$

which is $\Omega(|\mathrm{Adj}_{G^k}(i)|)$ if all variables are simple.[1] This degree computation is the most costly part of the minimum degree algorithm. When supervariables are present, the time taken is in the best case proportional to the degree of the variable in the "compressed" elimination graph, where all non-principal variables and their incident edges are removed.

4. Approximate degree. Having now discussed the data structures and the standard minimum degree implementations, we now consider our approximation for the minimum degree and indicate its lower complexity.

We assume that p is the kth pivot, and that we compute the bounds only for supervariables $\mathbf{i} \in \mathcal{L}_p$. Rather than computing the exact external degree, d_i, our *Approximate Minimum Degree* algorithm (AMD) computes an upper bound [7, 8],

$$\overline{d}_i^k = \min \left\{ \begin{array}{l} n-k, \\ \overline{d}_i^{k-1} + |\mathcal{L}_p \setminus \mathbf{i}|, \\ |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + \displaystyle\sum_{e \in \mathcal{E}_i \setminus \{p\}} |\mathcal{L}_e \setminus \mathcal{L}_p| \end{array} \right\}. \tag{4.1}$$

The first two terms ($n-k$, the size of the active submatrix, and $\overline{d}_i^{k-1} + |\mathcal{L}_p \setminus \mathbf{i}|$, the worst case fill-in) are usually not as tight as the third term in Equation (4.1). Algorithm 2 computes $|\mathcal{L}_e \setminus \mathcal{L}_p|$ for *all* elements e in the entire quotient graph. The set $\mathcal{L}_e$ splits into two disjoint subsets: the *external* subset $\mathcal{L}_e \setminus \mathcal{L}_p$ and the *internal* subset $\mathcal{L}_e \cap \mathcal{L}_p$. If Algorithm 2 scans element e, the term $w(e)$ is initialized to $|\mathcal{L}_e|$ and then decremented once for each variable i in the internal subset $\mathcal{L}_e \cap \mathcal{L}_p$, and, at the end of Algorithm 2, we have $w(e) = |\mathcal{L}_e| - |\mathcal{L}_e \cap \mathcal{L}_p| = |\mathcal{L}_e \setminus \mathcal{L}_p|$. If Algorithm 2 does not scan element e, the term $w(e)$ is less than zero. Combining these two cases, we obtain

$$|\mathcal{L}_e \setminus \mathcal{L}_p| = \left\{ \begin{array}{ll} w(e) & \text{if } w(e) \geq 0 \\ |\mathcal{L}_e| & \text{otherwise} \end{array} \right\}, \text{ for all } e \in \overline{V}. \tag{4.2}$$

[1] Asymptotic complexity notation is defined in [6]. We write $f(n) = \Theta(g(n))$ if there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n > n_0$. Similarly, $f(n) = \Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n > n_0$; and $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n > n_0$.

Algorithm 2 (Computation of $|\mathcal{L}_e \setminus \mathcal{L}_p|$ for all $e \in \overline{V}$)

```
assume w(1...n) < 0
for each supervariable i ∈ L_p do
    for each element e ∈ E_i do
        if (w(e) < 0) then w(e) = |L_e|
        w(e) = w(e) − |i|
    end for
end for
```

Algorithm 2 is followed by a second loop to compute our upper bound degree, $\overline{d}_i$ for each supervariable $\mathbf{i} \in \mathcal{L}_p$, using Equations (4.1) and (4.2). The total time for Algorithm 2 is

$$\Theta(\sum_{\mathbf{i} \in \mathcal{L}_p} |\mathcal{E}_i|).$$

The second loop to compute the upper bound degrees takes time

$$\Theta(\sum_{\mathbf{i} \in \mathcal{L}_p} (|\mathcal{A}_i| + |\mathcal{E}_i|)), \tag{4.3}$$

which is thus equal to the total asymptotic time.

Multiple elimination [25] improves the minimum degree algorithm by updating the degree of a variable only once for each set of independent pivots. Incomplete degree update [17, 18] skips the degree update of outmatched variables. We cannot take full advantage of the incomplete degree update since it avoids the degree update for some supervariables adjacent to the pivot element. With our technique (Algorithm 2), we must scan the element lists for all supervariables $\mathbf{i}$ in $\mathcal{L}_p$. If the degree update of one of the supervariables is to be skipped, its element list must still be scanned so that the external subset terms can be computed for the degree update of other supervariables in $\mathcal{L}_p$. The only advantage of multiple elimination or incomplete degree update would be to skip the second loop that computes the upper bound degrees for outmatched variables or supervariables for which the degree has already been computed.

If the total time in Equation (4.3) is amortized across the computation of all supervariables $\mathbf{i} \in \mathcal{L}_p$, then the time taken to compute $\overline{d}_i$ is

$$\Theta(|\mathcal{A}_i| + |\mathcal{E}_i|) = O(|\mathcal{A}_i^0|),$$

which is $\Theta(|\mathrm{Adj}_{\mathcal{G}^k}(i)|)$ if all variables are simple. Computing our bound takes time proportional to the degree of the variable in the *quotient graph*, $\mathcal{G}$. This is much faster than the time taken to compute the exact external degree (see Equation (3.3)).

4.1. Accuracy of our approximate degrees. Gilbert, Moler, and Schreiber [24] also use approximate external degrees that they can compute in the same time as our degree bound $\overline{d}$. In our notation, their bound $\widehat{d}_i$ is

$$\widehat{d}_i = |\mathcal{A}_i \setminus \mathbf{i}| + \sum_{e \in \mathcal{E}_i} |\mathcal{L}_e \setminus \mathbf{i}|.$$

Since many pivotal variables are adjacent to two or fewer elements when selected, Ashcraft and Eisenstat [4] have suggested a combination of $\widehat{d}$ and d,

$$\widetilde{d} = \begin{cases} d & \text{if } |\mathcal{E}_i| = 2 \\ \widehat{d} & \text{otherwise} \end{cases} .$$

Computing $\widetilde{d}$ takes the same time as $\overline{d}$ or $\widehat{d}$, except when $|\mathcal{E}_i| = 2$. In this case, it takes $O(|\mathcal{A}_i| + |\mathcal{L}_e|)$ time to compute $\widetilde{d}$, whereas computing $\overline{d}$ or $\widehat{d}$ takes $\Theta(|\mathcal{A}_i|)$ time. In the Yale Sparse Matrix Package [17] the $|\mathcal{L}_e \setminus \mathcal{L}_p|$ term for the $\mathcal{E}_i = \{e, p\}$ case is computed by scanning $\mathcal{L}_e$ once. It is then used to compute d_i for all $i \in \mathcal{L}_p$ for which $\mathcal{E}_i = \{e, p\}$. This technique can also be used to compute $\widetilde{d}$, and thus the time to compute $\widetilde{d}$ is $O(|\mathcal{A}_i| + |\mathcal{L}_e|)$ and not $\Theta(|\mathcal{A}_i| + |\mathcal{L}_e|)$.

Theorem 1: Relationship between external degree and the three approximate degree bounds. *The equality* $d_i = \overline{d}_i = \widetilde{d}_i = \widehat{d}_i$ *holds when* $|\mathcal{E}_i| \leq 1$. *The inequality* $d_i = \overline{d}_i = \widetilde{d}_i \leq \widehat{d}_i$ *holds when* $|\mathcal{E}_i| = 2$. *Finally, the inequality* $d_i \leq \overline{d}_i \leq \widetilde{d}_i = \widehat{d}_i$ *holds when* $|\mathcal{E}_i| > 2$.

Proof:

The bound $\widehat{d}_i$ is equal to the exact degree when variable i is adjacent to at most one element ($|\mathcal{E}_i| \leq 1$). The accuracy of their bound is unaffected by the size of $\mathcal{A}_i$, since entries are removed from $\mathcal{A}$ that fall within the pattern $\mathcal{L}$ of an element. Thus, if there is just one element (the current element p, say), the bound $\widehat{d}_i$ is tight. If $|\mathcal{E}_i|$ is two (the current element, p, and a prior element e, say), we have

$$\widehat{d}_i = |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + |\mathcal{L}_e \setminus \mathbf{i}| = d_i + |(\mathcal{L}_e \cap \mathcal{L}_p) \setminus \mathbf{i}|.$$

The bound $\widehat{d}_i$ counts entries in the set $(\mathcal{L}_e \cap \mathcal{L}_p) \setminus \mathbf{i}$ twice, and so $\widehat{d}_i$ will be an overestimate in the possible (even likely) case that a variable $j \neq i$ exists that is adjacent to both e and p. Combined with the definition of $\widetilde{d}$, we have $d_i = \widetilde{d}_i = \widehat{d}_i$ when $|\mathcal{E}_i| \leq 1$, $d_i = \widetilde{d}_i \leq \widehat{d}_i$ when $|\mathcal{E}_i| = 2$, and $d_i \leq \widetilde{d}_i = \widehat{d}_i$ when $|\mathcal{E}_i| > 2$.

If $|\mathcal{E}_i| \leq 1$ our bound $\overline{d}_i$ is exact for the same reason that $\widehat{d}_i$ is exact. If $|\mathcal{E}_i|$ is two we have

$$\overline{d}_i = |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + |\mathcal{L}_e \setminus \mathcal{L}_p| = d_i.$$

No entry is in both $\mathcal{A}_i$ and any element $\mathcal{L}$, since these redundant entries are removed from $\mathcal{A}_i$. Any entry in $\mathcal{L}_p$ does not appear in the external subset $(\mathcal{L}_e \setminus \mathcal{L}_p)$. Thus, no entry is counted twice, and $d_i = \overline{d}_i$ when $|\mathcal{E}_i| \leq 2$. Finally, consider both $\overline{d}_i$ and $\widehat{d}_i$ when $|\mathcal{E}_i| > 2$. We have

$$\overline{d}_i = |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + \sum_{e \in \mathcal{E}_i \setminus \{p\}} |\mathcal{L}_e \setminus \mathcal{L}_p|$$

and

$$\widehat{d}_i = |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + \sum_{e \in \mathcal{E}_i \setminus \{p\}} |\mathcal{L}_e \setminus \mathbf{i}|.$$

Since these degree bounds are only used when computing the degree of a supervariable $\mathbf{i} \in \mathcal{L}_p$, we have $\mathbf{i} \subseteq \mathcal{L}_p$. Thus, $\overline{d}_i \leq \widehat{d}_i$ when $|\mathcal{E}_i| > 2$.

□

Combining the three inequalities in Theorem 1, the inequality $d_i \le \overline{d}_i \le \widetilde{d}_i \le \widehat{d}_i$ holds for all values of $|\mathcal{E}_i|$.

Note that, if a variable i is adjacent to two elements or less then our bound is equal to the exact external degree. This is very important, since most variables of minimum degree are adjacent to two elements or less. Additionally, our degree bounds take advantage of element absorption, since the bound depends on $|\mathcal{E}_i|$ after elements are absorbed.

4.2. Degree computation example. We illustrate the computation of our approximate external degree bound in Figures 3.1 and 3.2. Variable 6 is adjacent to three elements in $\mathcal{G}^3$ and $\mathcal{G}^4$. All other variables are adjacent to two or less elements. In $\mathcal{G}^3$, the bound $\overline{d}_6$ is tight, since the two sets $|\mathcal{L}_1 \setminus \mathcal{L}_3|$ and $|\mathcal{L}_2 \setminus \mathcal{L}_3|$ are disjoint.

In graph $\mathcal{G}^4$, the current pivot element is $p = 4$. We compute

$$\begin{aligned}
\overline{d}_6 &= |\mathcal{A}_i \setminus \mathbf{i}| + |\mathcal{L}_p \setminus \mathbf{i}| + (\sum_{e \in \mathcal{E}_i \setminus \{p\}} |\mathcal{L}_e \setminus \mathcal{L}_p|) \\
&= |\emptyset \setminus \{6\}| + |\{6,7,8\} \setminus \{6\}| + (|\mathcal{L}_2 \setminus \mathcal{L}_4| + |\mathcal{L}_3 \setminus \mathcal{L}_4|) \\
&= |\{7,8\}| + (|\{5,6,9\} \setminus \{6,7,8\}| + |\{5,6,7\} \setminus \{6,7,8\}|) \\
&= |\{7,8\}| + (|\{5,9\}| + |\{5\}|) \\
&= 5.
\end{aligned}$$

The exact external degree of variable 6 is $d_6 = 4$, as can be seen in the elimination graph G^4 on the left of Figure 3.2(a). Our bound is one more than the exact external degree, since the variable 5 appears in both $\mathcal{L}_2 \setminus \mathcal{L}_4$ and $\mathcal{L}_3 \setminus \mathcal{L}_4$, but is one less than the bound $\widehat{d}_i$ which is equal to 6 in this case. Our bound on the degree of variable 6 is again tight after the next pivot step, since elements 2 and 3 are absorbed into element 5.

5. The approximate minimum degree algorithm. The Approximate Minimum Degree algorithm is identical to Algorithm 1, except that the external degree, d_i, is replaced with $\overline{d}_i$, throughout. The bound on the external degree, $\overline{d}_i$, is computed using Algorithm 2 and Equations (4.1) and (4.2). In addition to absorbing elements in $\mathcal{E}_p$, any element with an empty external subset ($|\mathcal{L}_e \setminus \mathcal{L}_p| = 0$) is also absorbed into element p, even if e is not adjacent to p. This "aggressive" element absorption improves the degree bounds by reducing $|\mathcal{E}|$.

As in many other minimum degree algorithms, we use linked lists to assist the search for a variable of minimum degree. List d holds all supervariables $\mathbf{i}$ with degree bound $\overline{d}_i = d$. Maintaining this data structure takes time proportional to the total number of degree computations, or $O(|\mathbf{L}|)$.

Computing the pattern of each pivot element, $\mathcal{L}_p$, takes a total of $O(|\mathbf{L}|)$ time overall, since each element is used in the computation of at most one other element, and the total sizes of all elements constructed is $O(|\mathbf{L}|)$.

The AMD algorithm is based on the quotient graph data structure used in the MA27 minimum degree algorithm [13, 14, 15]. Initially, the sets $\mathcal{A}$ are stored, followed by a small amount of elbow room. When the set $\mathcal{L}_p$ is formed, it is placed in the elbow room (or in place of $\mathcal{A}_p$ if $|\mathcal{E}_p| = 0$). Garbage collection occurs if the elbow room is exhausted. During garbage collection, the space taken by $\mathcal{A}_i$ and $\mathcal{E}_i$ is reduced to exactly $|\mathcal{A}_i| + |\mathcal{E}_i|$ for each supervariable $\mathbf{i}$ (which is less than or equal to $|\mathcal{A}_i^0|$) and the extra space is reclaimed. The space for $\mathcal{A}_e$ and $\mathcal{E}_e$ for all elements $e \in \overline{V}$ is fully reclaimed, as is the space for $\mathcal{L}_e$ of any absorbed elements e. Each garbage collection

takes time that is proportional to the size of the workspace (normally $\Theta(|\mathbf{A}|)$). In practice, elbow room of size n is sufficient.

During the computation of our degree bounds, we compute the following hash function for supervariable detection [3],

$$\text{Hash}(i) = \left\{ \left(\sum_{\mathbf{j} \in \mathcal{A}_i} j + \sum_{e \in \mathcal{E}_i} e \right) \bmod (n-1) \right\} + 1,$$

which increases the degree computation time by a small constant factor. We place each supervariable $\mathbf{i}$ in a hash bucket according to Hash(i), taking time $O(|\mathbf{L}|)$ overall. If two or more supervariables are placed in the same hash bucket, then each pair of supervariables $\mathbf{i}$ and $\mathbf{j}$ in the hash bucket are tested for indistinguishability. If the hash function results in no collisions then the total time taken by the comparison is $O(|\mathbf{A}|)$.

Ashcraft [3] uses this hash function as a preprocessing step on the entire matrix (without the mod$(n-1)$ term, and with an $O(|V| \log |V|)$ sort instead of $|V|$ hash buckets). In contrast, we use this function during the ordering, and only hash those variables adjacent to the current pivot element.

For example, variables 7, 8, and 9 are indistinguishable in G^5, in Figure 3.2(a). The AMD algorithm would not consider variable 8 at step 5, since it is not adjacent to the pivot element 5 (refer to quotient graph $\mathcal{G}^5$ in Figure 3.2(b)). AMD would not construct $\mathbf{7} = \{7, 9\}$ at step 5, since 7 and 9 are distinguishable in $\mathcal{G}^5$. It would construct $\mathbf{7} = \{7, 8, 9\}$ at step 6, however.

The total number of times the approximate degree $\overline{d}_i$ of variable i is computed during elimination is no more than the number of nonzero entries in row k of $\mathbf{L}$, where variable i is the kth pivot. The time taken to compute $\overline{d}_i$ is $O(|\mathcal{A}_i^0|)$, or equivalently $O(|(\mathbf{PAP}^T)_{k*}|)$, the number of nonzero entries in row k of the permuted matrix. The total time taken by the entire AMD algorithm is thus bounded by the degree computation,

$$O\left(\sum_{k=1}^{n} |\mathbf{L}_{k*}| \cdot |(\mathbf{PAP}^T)_{k*}| \right). \tag{5.1}$$

This bound assumes no (or few) supervariable hash collisions and a constant number of garbage collections. In practice these assumptions seem to hold, but the asymptotic time would be higher if they did not. In many problem domains, the number of nonzeros per row of $\mathbf{A}$ is a constant, independent of n. For matrices in these domains, our AMD algorithm takes time $O(|\mathbf{L}|)$ (with the same assumptions).

6. Performance results. In this section, we present the results of our experiments with AMD on a wide range of test matrices. We first compare the degree computations discussed above (t, d, $\overline{d}$, $\widetilde{d}$, and $\widehat{d}$), as well as an upper bound on the true degree, $\overline{t} \equiv \overline{d} + |\mathbf{i}| - 1$. We then compare the AMD algorithm with other established minimum degree codes (MMD and MA27).

6.1. Test Matrices. We tested all degree bounds and codes on all matrices in the Harwell/Boeing collection of type PUA, RUA, PSA, and RSA [11, 12] (at `orion.cerfacs.fr` or `numerical.cc.rl.ac.uk`), all non-singular matrices in Saad's SPARSKIT2 collection (at `ftp.cs.umn.edu`), all matrices in the University of Florida collection (available from `ftp.cis.ufl.edu` in the directory `pub/umfpack/matrices`),

TABLE 6.1
Selected matrices in test set

Matrix	n	nz	Percentage of $\lvert\mathcal{E}_p\rvert > 2$	Percentage of $\lvert\mathcal{E}_i\rvert > 2$	Description
RAEFSKY3	21,200	733,784	0.00	13.4	fluid/structure interaction, turbulence
VENKAT01	62,424	827,684	0.71	15.7	unstructured 2D Euler solver
BCSSTK32	44,609	985,046	0.20	27.3	structural eng., automobile chassis
EX19	12,005	123,937	1.57	29.4	2D developing pipe flow (turbulent)
BCSSTK30	28,924	1,007,284	0.66	31.8	structural eng., off-shore platform
CT20STIF	52,329	1,323,067	0.77	33.2	structural eng., CT20 engine block
NASASRB	54,870	1,311,227	0.06	35.0	shuttle rocket booster
OLAF	16,146	499,505	0.41	35.2	NASA test problem
RAEFSKY1	3,242	145,517	0.00	38.9	incompressible flow, pressure-driven pipe
CRYSTK03	24,696	863,241	0.00	40.9	structural eng., crystal vibration
RAEFSKY4	19,779	654,416	0.00	41.4	buckling problem for container model
CRYSTK02	13,965	477,309	0.00	42.0	structural eng., crystal vibration
BCSSTK33	8,738	291,583	0.00	42.6	structural eng., auto steering mech.
BCSSTK31	35,588	572,914	0.60	43.1	structural eng., automobile component
EX11	16,614	540,167	0.04	43.3	CFD, 3D cylinder & flat plate heat exch.
FINAN512	74,752	261,120	1.32	46.6	economics, portfolio optimization
RIM	22,560	862,411	2.34	63.2	chemical eng., fluid mechanics problem
BBMAT	38,744	1,274,141	5.81	64.4	CFD, 2D airfoil with turbulence
EX40	7,740	225,136	17.45	64.7	CFD, 3D die swell problem on square die
WANG4	26,068	75,564	15.32	78.3	3D MOSFET semicond. (30x30x30 grid)
LHR34	35,152	608,830	7.69	78.7	chemical eng., light hydrocarbon recovery
WANG3	26,064	75,552	15.29	79.2	3D diode semiconductor (30x30x30 grid)
LHR71	70,304	1,199,704	8.47	81.1	chemical eng., light hydrocarbon recovery
ORANI678	2,529	85,426	6.68	86.9	Australian economic model
PSMIGR1	3,140	410,781	6.65	91.0	US county-by-county migration
APPU	14,000	1,789,392	15.64	94.4	NASA test problem (random matrix)

and several other matrices from NASA and Boeing. Of those 378 matrices, we present results below on those matrices requiring 500 million or more floating-point operations for the Cholesky factorization, as well as the ORANI678 matrix in the Harwell/Boeing collection and the EX19 in Saad's collection (a total of 26 matrices). The latter two are best-case and worst-case examples from the set of smaller matrices.

For the unsymmetric matrices in the test set, we first used the maximum transversal algorithm `MC21` from the Harwell Subroutine Library [9] to reorder the matrix so that the permuted matrix has a zero-free diagonal. We then formed the symmetric pattern of the permuted matrix plus its transpose. This is how a minimum degree ordering algorithm is used in MUPS [1, 2]. For these matrices, Table 6.1 lists the statistics for the symmetrized pattern.

Table 6.1 lists the matrix name, the order, the number of nonzeros in lower triangular part, two statistics obtained with an exact minimum degree ordering (using d), and a description. In column 4, we report the percentage of pivots p such that $|\mathcal{E}_p| > 2$. Column 4 shows that there is only a small percentage of pivots selected using an exact minimum degree ordering that have more than two elements in their adjacency list. Therefore, we can expect a good quality ordering with an algorithm based on our approximate degree bound. In column 5, we indicate how often a degree d_i is computed when $|\mathcal{E}_i| > 2$ (as a percentage of the total number of degree updates). Table 6.1 is sorted according to this degree update percentage. Column 5 thus reports the percentage of "costly" degree updates performed by a minimum degree algorithm based on the exact degree. For matrices with relatively large values

in column 5, significant time reductions can be expected with an approximate degree based algorithm.

Since any minimum degree algorithm is sensitive to tie-breaking issues, we randomly permuted all matrices and their adjacency lists 21 times (except for the random APPU matrix, which we ran only once). All methods were given the same set of 21 randomized matrices. We also ran each method on the original matrix. On some matrices, the original matrix gives better ordering time and fill-in results for all methods than the best result obtained with the randomized matrices. The overall comparisons are not however dependent on whether original or randomized matrices are used. We thus report only the median ordering time and fill-in obtained for the randomized matrices.

The APPU matrix is a random matrix used in a NASA benchmark, and is thus not representative of sparse matrices from real problems. We include it in our test set as a pathological case that demonstrates how well AMD handles a very irregular problem. Its factors are about 90% dense. It was not practical to run the APPU matrix 21 times because the exact degree update algorithms took too much time.

6.2. Comparing the exact and approximate degrees. To make a valid comparison between degree update methods, we modified our code for the AMD algorithm so that we could compute the exact external degree (d), our bound ($\overline{d}$), Ashcraft and Eisenstat's bound ($\widetilde{d}$), Gilbert, Moler, and Schreiber's bound ($\widehat{d}$), the exact true degree (t), and our upper bound on the true degree ($\overline{t}$). The six codes based on d, $\overline{d}$, $\widetilde{d}$, $\widehat{d}$, t, and $\overline{t}$ (columns 3 to 8 of Table 2) differ only in how they compute the degree. Since aggressive absorption is more difficult when using some bounds than others, we switched off aggressive absorption for these six codes. The actual AMD code (in column 2 of Table 2) uses $\overline{d}$ with aggressive absorption.

Table 6.2 lists the median number of nonzeros below the diagonal in $\mathbf{L}$ (in thousands) for each method. Results 20% higher than the lowest median $|\mathbf{L}|$ in the table (or higher) are underlined. Our upper bound on the true degree ($\overline{t}$) and the exact true degree (t) give nearly identical results. As expected, using minimum degree algorithms based on external degree noticeably improves the quality of the ordering (compare columns 3 and 7, or columns 4 and 8). From the inequality $d \leq \overline{d} \leq \widetilde{d} \leq \widehat{d}$, we would expect a similar ranking in the quality of ordering produced by these methods. Table 6.2 confirms this. The bound $\overline{d}$ and the exact external degree d produce nearly identical results. Comparing the AMD results and the $\overline{d}$ column, aggressive absorption tends to result in slightly lower fill-in, since it reduces $|\mathcal{E}|$ and thus improves the accuracy of our bound. The $\widetilde{d}$ bound is often accurate enough to produce good results, but can fail catastrophically for matrices with a high percentage of approximate pivots (see column 4 in Table 6.1). The less accurate $\widehat{d}$ bound produces notably worse results for many matrices.

Comparing all 378 matrices, the median $|\mathbf{L}|$ when using $\overline{d}$ is never more than 9% higher than the median fill-in obtained when using the exact external degree, d (with the exception of the FINAN512 matrix). The fill-in results for d and $\overline{d}$ are identical for nearly half of the 378 matrices. The approximate degree bound $\overline{d}$ thus gives a very reliable estimation of the degree in the context of a minimum degree algorithm.

The FINAN512 matrix is highly sensitive to tie-breaking variations. Its graph consists of two types of nodes: "constraint" nodes and "linking" nodes [5]. The constraint nodes form independent sparse subgraphs, connected together via a tree of linking nodes. This matrix is a pathological worst-case matrix for any minimum degree

TABLE 6.2
Median fill-in results of the degree update methods

Matrix	Number of nonzeros below diagonal in **L**, in thousands						
	AMD	d	$\overline{d}$	$\widetilde{d}$	$\widehat{d}$	t	$\overline{t}$
RAEFSKY3	4709	4709	4709	4709	5114	4992	4992
VENKAT01	5789	5771	5789	5798	6399	6245	6261
BCSSTK32	5080	5081	5079	5083	5721	5693	5665
EX19	319	319	319	318	366	343	343
BCSSTK30	3752	3751	3753	3759	4332	4483	4502
CT20STIF	10858	10758	10801	11057	13367	12877	12846
NASASRB	12282	12306	12284	12676	14909	14348	14227
OLAF	2860	2858	2860	2860	3271	3089	3090
RAEFSKY1	1151	1151	1151	1151	1318	1262	1262
CRYSTK03	13836	13836	13836	13836	17550	15507	15507
RAEFSKY4	7685	7685	7685	7685	9294	8196	8196
CRYSTK02	6007	6007	6007	6007	7366	6449	6449
BCSSTK33	2624	2624	2624	2640	3236	2788	2787
BCSSTK31	5115	5096	5132	5225	6194	6079	6057
EX11	6014	6016	6014	6014	7619	6673	6721
FINAN512	4778	4036	6042	11418	11505	8235	8486
RIM	3948	3898	3952	3955	4645	4268	4210
BBMAT	19673	19880	19673	21422	37820	21197	21445
EX40	1418	1386	1417	1687	1966	1526	1530
WANG4	6547	6808	6548	6566	7871	7779	7598
LHR34	3618	3743	3879	11909	27125	4383	4435
WANG3	6545	6697	6545	6497	7896	7555	7358
LHR71	7933	8127	8499	28241	60175	9437	9623
ORANI678	147	147	146	150	150	147	146
PSMIGR1	3020	3025	3011	3031	3176	2966	2975
APPU	87648	87613	87648	87566	87562	87605	87631

method. All constraint nodes should be ordered first, but linking nodes have low degree and tend to be selected first, which causes high fill-in. Using a tree dissection algorithm, Berger, Mulvey, Rothberg, and Vanderbei [5] obtain an ordering with only 1.83 million nonzeros in **L**.

Table 6.3 lists the median ordering time (in seconds on a SUN SPARCstation 10) for each method. Ordering time twice that of the minimum median ordering time listed in the table (or higher) is underlined. Computing the $\widehat{d}$ bound is often the fastest, since it requires a single pass over the element lists instead of the two passes required for the $\overline{d}$ bound. It is, however, sometimes slower than $\overline{d}$ because it can generate more fill-in, which increases the ordering time (see Equation 5.1). The ordering time of the two exact degree updates (d and t) increases dramatically as the percentage of "costly" degree updates increases (those for which $|\mathcal{E}_i| > 2$).

Garbage collection has little effect on the ordering time obtained. In the above runs, we gave each method elbow room of size n. Usually a single garbage collection occurred. At most two garbage collections occurred for AMD, and at most three for the other methods (aggressive absorption reduces the memory requirements).

6.3. Comparing algorithms. In this section, we compare AMD with two other established minimum degree codes: Liu's Multiple Minimum Degree (MMD) code [25] and Duff and Reid's MA27 code [15]. MMD stores the element patterns $\mathcal{L}$ in a fragmented manner and requires no elbow room [20, 21]. It uses the exact external degree, d. MMD creates supervariables only when two variables **i** and **j** have no

TABLE 6.3
Median ordering time of the degree update methods

Matrix	Ordering time, in seconds						
	AMD	d	$\overline{d}$	$\widetilde{d}$	$\widehat{d}$	t	$\overline{t}$
RAEFSKY3	1.05	1.10	1.09	1.05	1.02	1.15	1.09
VENKAT01	4.07	4.95	4.11	4.47	3.88	4.32	3.85
BCSSTK32	4.67	5.64	4.54	4.91	4.35	5.55	4.48
EX19	0.87	1.12	0.89	1.01	0.86	1.09	0.87
BCSSTK30	3.51	5.30	3.55	3.65	3.51	4.38	3.38
CT20STIF	6.62	8.66	6.54	7.07	6.31	8.63	6.45
NASASRB	7.69	11.03	7.73	9.23	7.78	11.78	7.99
OLAF	1.83	2.56	1.90	2.16	1.83	2.33	1.78
RAEFSKY1	0.27	0.34	0.28	0.32	0.25	0.35	0.28
CRYSTK03	3.30	4.84	3.08	3.68	3.14	5.23	3.30
RAEFSKY4	2.32	2.90	2.18	2.45	2.08	3.12	2.07
CRYSTK02	1.49	2.34	1.55	1.64	1.45	2.04	1.52
BCSSTK33	0.91	1.36	1.05	0.99	0.85	1.62	0.91
BCSSTK31	4.55	7.53	4.92	5.68	4.56	7.41	4.92
EX11	2.70	4.06	2.77	3.00	2.60	4.23	2.89
FINAN512	15.03	34.11	14.45	17.79	15.84	46.49	18.58
RIM	5.74	10.38	5.69	6.12	5.72	10.01	5.58
BBMAT	27.80	115.75	27.44	42.17	23.02	129.32	28.33
EX40	1.04	1.56	1.10	1.09	0.95	1.46	1.12
WANG4	5.45	11.45	5.56	6.98	5.21	11.59	5.88
LHR34	19.56	109.10	25.62	45.36	43.70	125.41	24.73
WANG3	5.02	10.45	5.49	6.52	4.81	11.02	5.02
LHR71	46.03	349.58	58.25	129.85	121.96	389.70	60.40
ORANI678	5.49	196.01	8.13	6.97	7.23	199.01	8.45
PSMIGR1	10.61	334.27	10.07	14.20	8.16	339.28	9.94
APPU	41.75	2970.54	39.83	43.20	40.64	3074.44	38.93

adjacent variables and exactly two adjacent elements ($\mathcal{E}_i = \mathcal{E}_j = \{e, p\}$, and $\mathcal{A}_i = \mathcal{A}_j = \emptyset$, where p is the current pivot element). A hash function is not required. MMD takes advantage of multiple elimination and incomplete update.

MA27 uses the true degree, t, and the same data structures as AMD. It detects supervariables whenever two variables are adjacent to the current pivot element and have the same structure in the quotient graph (as does AMD). MA27 uses the true degree as the hash function for supervariable detection, and does aggressive absorption. Neither AMD nor MA27 take advantage of multiple elimination or incomplete update.

Structural engineering matrices tend to have many rows of identical nonzero pattern. Ashcraft has found that the total ordering time of MMD can be significantly improved by detecting these initial supervariables before starting the elimination [3]. We implemented Ashcraft's pre-compression algorithm, and modified MMD to allow for initial supervariables. We call the resulting code CMMD ("compressed" MMD). Pre-compression has little effect on AMD, since it finds these supervariables when their degrees are first updated. The AMD algorithm on compressed matrices together with the cost of pre-compression was never faster than AMD.

Table 6.4 lists the median number of nonzeros below the diagonal in $\mathbf{L}$ (in thousands) for each code. Results 20% higher than the lowest median $|\mathbf{L}|$ in the table (or higher) are underlined. AMD, MMD, and CMMD find orderings of about the same quality. MA27 is slightly worse because it uses the true degree (t) instead of the external degree (d).

TABLE 6.4
Median fill-in results of the four codes

Matrix	Number of nonzeros below diagonal in **L**, in thousands			
	AMD	MMD	CMMD	MA27
RAEFSKY3	4709	4779	4724	5041
VENKAT01	5789	5768	5811	6303
BCSSTK32	5080	5157	5154	5710
EX19	319	322	324	345
BCSSTK30	3752	3788	3712	4529
CT20STIF	10858	11212	10833	12760
NASASRB	12282	12490	12483	14068
OLAF	2860	2876	2872	3063
RAEFSKY1	1151	1165	1177	1255
CRYSTK03	13836	13812	14066	15496
RAEFSKY4	7685	7539	7582	8245
CRYSTK02	6007	5980	6155	6507
BCSSTK33	2624	2599	2604	2766
BCSSTK31	5115	5231	5216	6056
EX11	6014	5947	6022	6619
FINAN512	4778	8180	8180	8159
RIM	3948	3947	3914	4283
BBMAT	19673	19876	19876	21139
EX40	1418	1408	1401	1521
WANG4	6547	6619	6619	7598
LHR34	3618	4162	4162	4384
WANG3	6545	6657	6657	7707
LHR71	7933	9190	9190	9438
ORANI678	147	147	147	147
PSMIGR1	3020	2974	2974	2966
APPU	87648	87647	87647	87605

Considering the entire set of 378 matrices, AMD produces a better median fill-in than MMD, CMMD, and MA27 for 62%, 61%, and 81% of the matrices, respectively. AMD never generates more than 7%, 7%, and 4% more nonzeros in **L** than MMD, CMMD, and MA27, respectively. We have shown empirically that AMD produces an ordering at least as good as these other three methods for this large test set.

If the apparent slight difference in ordering quality between AMD and MMD is statistically significant, we conjecture that it has more to do with earlier supervariable detection (which affects the external degree) than with the differences between the external degree and our upper bound.

Table 6.5 lists the median ordering time (in seconds on a SUN SPARCstation 10) for each method. The ordering time for CMMD includes the time taken by the pre-compression algorithm. Ordering time twice that of the minimum median ordering time listed in the table (or higher) is underlined. On certain classes of matrices, typically those from structural analysis applications, CMMD is significantly faster than MMD. AMD is the fastest method for all but the EX19 matrix. For the other 352 matrices in our full test set, the differences in ordering time between these various methods is typically less. If we compare the ordering time of AMD with the other methods on all matrices in our test set requiring at least a tenth of a second of ordering time, then AMD is slower than MMD, CMMD, and MA27 only for 6, 15, and 8 matrices respectively. For the full set of matrices, AMD is never more than 30% slower than these other methods. The best and worst cases for the relative run-time

TABLE 6.5
Median ordering time of the four codes

Matrix	Ordering time, in seconds			
	AMD	MMD	CMMD	MA27
RAEFSKY3	1.05	2.79	1.18	1.23
VENKAT01	4.07	9.01	4.50	5.08
BCSSTK32	4.67	12.47	5.51	6.21
EX19	0.87	0.69	0.83	1.03
BCSSTK30	3.51	7.78	3.71	4.40
CT20STIF	6.62	26.00	9.59	9.81
NASASRB	7.69	22.47	11.28	12.75
OLAF	1.83	5.67	4.41	2.64
RAEFSKY1	0.27	0.82	0.28	0.40
CRYSTK03	3.30	10.63	3.86	5.27
RAEFSKY4	2.32	5.24	2.36	2.91
CRYSTK02	1.49	3.89	1.53	2.37
BCSSTK33	0.91	2.24	1.32	1.31
BCSSTK31	4.55	11.60	7.76	7.92
EX11	2.70	7.45	5.05	3.90
FINAN512	15.03	895.23	897.15	40.31
RIM	5.74	9.09	8.11	10.13
BBMAT	27.80	200.86	201.03	134.58
EX40	1.04	2.13	2.04	1.30
WANG4	5.45	10.79	11.60	9.86
LHR34	19.56	139.49	141.16	77.83
WANG3	5.02	10.37	10.62	8.23
LHR71	46.03	396.03	400.40	215.01
ORANI678	5.49	124.99	127.10	124.66
PSMIGR1	10.61	186.07	185.74	229.51
APPU	41.75	5423.23	5339.24	2683.27

of AMD for the smaller matrices are included in Table 6.5 (the EX19 and ORANI678 matrices).

7. Summary. We have described a new upper bound for the degree of nodes in the elimination graph that can be easily computed in the context of a minimum degree algorithm. We have demonstrated that this upper-bound for the degree is more accurate than all previously used degree approximations. We have experimentally shown that we can replace an exact degree update by our approximate degree update and obtain almost identical fill-in.

An Approximate Minimum Degree (AMD) based on external degree approximation has been described. We have shown that the AMD algorithm is highly competitive with other ordering algorithms. It is typically faster than other minimum degree algorithms, and produces comparable results to MMD (which is also based on external degree) in terms of fill-in. AMD typically produces better results, in terms of fill-in and computing time, than the MA27 minimum degree algorithm (based on true degrees).

8. Acknowledgments. We would like to thank John Gilbert for outlining the $\overline{d}_i \leq \widehat{d}_i$ portion of the proof to Theorem 1, Joseph Liu for providing a copy of the MMD algorithm, and Cleve Ashcraft and Stan Eisenstat for their comments on a draft of this paper.

REFERENCES

[1] P. R. Amestoy, *Factorization of large sparse matrices based on a multifrontal approach in a multiprocessor environment*, INPT PhD Thesis TH/PA/91/2, CERFACS, Toulouse, France, 1991.

[2] P. R. Amestoy, M. Daydé, and I. S. Duff, *Use of level 3 BLAS in the solution of full and sparse linear equations*, in High Performance Computing: Proceedings of the International Symposium on High Performance Computing, Montpellier, France, 22–24 March, 1989, J.-L. Delhaye and E. Gelenbe, eds., Amsterdam, 1989, North Holland, pp. 19–31.

[3] C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM Journal on Scientific Computing, (1995, to appear).

[4] C. Ashcraft and S. C. Eisenstat. personal communication.

[5] A. Berger, J. Mulvey, E. Rothberg, and R. Vanderbei, *Solving multistage stochachastic programs using tree dissection*, Tech. Report SOR-97-07, Program in Statistics and Operations Research, Princeton University, Princeton, New Jersey, 1995.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusets, and McGraw-Hill, New York, 1990.

[7] T. A. Davis and I. S. Duff, *Unsymmetric-pattern multifrontal methods for parallel sparse LU factorization*, Tech. Report TR-91-023, CIS Dept., Univ. of Florida, Gainesville, FL, 1991.

[8] ———, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, Tech. Report TR-94-038, CIS Dept., Univ. of Florida, Gainesville, FL, 1994. (submitted to the SIAM Journal on Matrix Analysis and Applications in March 1993, revised).

[9] I. S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Transactions on Mathematical Software, 7 (1981), pp. 315–330.

[10] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, London: Oxford Univ. Press, 1986.

[11] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15 (1989), pp. 1–14.

[12] ———, *Users' guide for the Harwell-Boeing sparse matrix collection (Release 1)*, Tech. Report RAL-92-086, Rutherford Appleton Laboratory, Didcot, Oxon, England, Dec. 1992.

[13] I. S. Duff and J. K. Reid, *A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination*, Journal of the Institute of Mathematics and its Applications, 14 (1974), pp. 281–291.

[14] ———, *MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Tech. Report AERE R10533, HMSO, London, 1982.

[15] ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Softw., 9 (1983), pp. 302–325.

[16] ———, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 633–641.

[17] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *Yale sparse matrix package, I: The symmetric codes*, International Journal for Numerical Methods in Engineering, 18 (1982), pp. 1145–1151.

[18] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM Journal on Scientific and Statistical Computing, 2 (1981), pp. 225–237.

[19] A. George and J. W. H. Liu, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Transactions on Mathematical Software, 6 (1980), pp. 337–358.

[20] ———, *A minimal storage implementation of the minimum degree algorithm*, SIAM J. Numer. Anal., 17 (1980), pp. 282–299.

[21] ———, *Computer Solution of Large Sparse Positive Definite Systems*, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[22] ———, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[23] A. George and D. R. McIntyre, *On the application of the minimum degree algorithm to finite element systems*, SIAM J. Numer. Anal., 15 (1978), pp. 90–111.

[24] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

[25] J. W. H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Softw., 11 (1985), pp. 141–153.

[26] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.

[27] D. J. Rose, *Symmetric Elimination on Sparse Positive Definite Systems and the Potential Flow Network Problem*, PhD thesis, Applied Math., Harvard Univ., 1970.

[28] ———, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., New York: Academic Press, 1973, pp. 183–217.

[29] B. SPEELPENNING, *The generalized element method*, Tech. Report Technical Report UIUCDCS-R-78-946, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1978.

[30] W. F. TINNEY AND J. W. WALKER, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. of the IEEE, 55 (1967), pp. 1801–1809.

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

Note: all University of Florida technical reports in this list of references are available in postscript form via anonymous ftp to `ftp.cis.ufl.edu` in the directory `cis/tech-reports`.