The Computational Complexity of the Minimum Degree Algorithm *

P. Heggernes[†] S. C. Eisenstat[‡] G. Kumfert[§] A. Pothen[¶]

Abstract

The Minimum Degree algorithm, one of the classical algorithms of sparse matrix computations, is widely used to order graphs to reduce the work and storage needed to solve sparse systems of linear equations. There has been extensive research involving practical implementations of this algorithm over the past two decades. However, little has been done to establish theoretical bounds on the computational complexity of these implementations. We study the Minimum Degree algorithm, and prove time complexity bounds for its widely used variants.

1 Introduction and motivation

One of the most famous and well studied problems of graph theory is the problem of adding as few edges as possible to a given graph so that the resulting graph is chordal. This is called the minimum fill problem, and it has applications in many areas within computer science, especially in sparse matrix computations [5, 9, 10, 11, 12]. As the minimum fill problem is NP-hard [14], several heuristics have been proposed to find low fill. One of the most famous and widely used of these heuristics is the Minimum Degree (MD) algorithm [6, 8, 13].

One rigid requirement of a practical MD implementation is that its space complexity is linear in the size of the input graph. Several algorithmic variants of the MD algorithm have been developed since it was first proposed in 1957, and these enhancements reduce the running time of the algorithm or reduce the fill generated by the ordering. However, the theoretical time complexity of the practical MD algorithm has never been established. Now that the increasing power of modern

*This work was supported by the National Science Foundation grant DMS-9807172, by the Department of Energy under subcontract B347882 from the Lawrence Livermore National Laboratory, and by NASA under contract NAS1-97046 while the last author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-2199. This research was initiated while the first author was visiting Old Dominion University in June 2000.

[†]Department of Informatics, University of Bergen, NO-5020 Bergen, Norway. pinar.heggernes@ii.uib.no

[‡]Department of Computer Science, Yale University, New Haven, CT 06520-2825 USA. stanley.eisenstat@yale.edu

[§]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, CA 94551-0808 USA. kumfert@llnl.gov

[¶]Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162 USA. pothen@cs.odu.edu and ICASE, NASA Langley Research Center, Hampton, VA 23681-2199 USA. pothen@icase.edu. A part of this work was done while the author was at the Computer Science Research Institute at Sandia National Labs, Albuquerque, NM.

microprocessors enable us to order very large graphs (with millions of vertices), the asymptotic bounds obtained from the theoretical analysis could be met on some large worst-case examples. Our aim in this paper is to study the MD algorithm, explaining the steps in its modern implementation, and to give a theoretical time bound on its running time. We will also show with an example that the presented time bound is tight on general graphs.

This paper is organized as follows: We provide the necessary graph theoretical background in Section 2. In Section 3, the various MD algorithms are described and their time complexity is analyzed, along with examples on which the bounds are attained. We conclude in Section 4.

2 Graph elimination and fill

A graph $G = (V, E)$ consists of a set V of *vertices* (or *nodes*), and a set $E \subseteq V \times V$ of *edges*. Vertices u and v are *adjacent*, or *neighbors*, if (u, v) is an edge in E. An *ordering* $\alpha : V \leftrightarrow \{1, 2, ..., n\}$ of G is a permutation, or a numbering, of its vertices; here $n \equiv |V|$. The graph G ordered by α is denoted by G_α, however we will omit the subscript when the ordering is clear from the context. If the vertices of G are ordered already, we will write $V = \{1, 2, ..., n\}$. The set of vertices adjacent to a vertex i in G_α is denoted by $adj_G(i)$. The *degree* of i in G is $d_G(i) = |adj_G(i)|$. For a set of vertices $X \subset V$, $adj(X) = \cup_{i \in X} adj(i) - X$, and the *external degree* of X is $|adj(X)|$. A set K of vertices is an *independent set* if no two vertices of K are adjacent. A set C of vertices is a *clique* if every pair of vertices in C is adjacent.

A *chord* in a cycle is an edge that connects two non-consecutive vertices of the cycle. A graph is *chordal* if every cycle with more than three edges contains a chord.

2.1 Elimination graph model

A graph model of the Cholesky factorization of a sparse matrix A is given in the algorithm [9] shown in Figure 1. This algorithm is often referred to as *the elimination game*.

$G_0 = G$;
for $i = 1$ **to** n **do**
  Add edges as necessary to make all neighbors of vertex i in G_{i-1} pairwise adjacent;
  Remove the vertex i and all edges incident to i;
  Denote the resulting graph by G_i;

Figure 1: The elimination game.

The input to the elimination game is $G = G(A)$. Before elimination, we assume an ordering on the vertices of G. At each step i, the neighborhood of vertex i is turned into a clique, and i is deleted from the graph. This is referred to as *eliminating* vertex i, and the graphs $G_i = (\{i + 1, ..., n\}, E_i)$ are called *elimination graphs*. (The set E_i contains the edges in the ith elimination graph G_i.) The *filled graph* $G^+ = (V, E^+)$ is obtained by adding to G all the edges added by the algorithm. Thus $E^+ = \cup_{i=0}^{n-1} E_i$, and the set of *fill edges* is $F = E^+ \setminus E$. We will let $m \equiv |E|$ and $m^+ \equiv |E^+|$.

Fulkerson and Gross [3] showed that the filled graphs resulting from this algorithm are exactly the class of chordal graphs. Different filled graphs result from processing the vertices of G in different orders. Thus in order to find a low fill, it is important to find a good order on the vertices of the given graph before running elimination game. Finding an ordering that results in the minimum fill is an NP-hard problem [14].

2.2 The minimum degree idea

The minimum degree idea aims to minimize fill locally at each step i of the elimination game by choosing to eliminate a vertex with the minimum degree in the elimination graph G_{i-1}. The algorithm starts by assuming that there is no numbering on the vertices, and chooses a vertex in G with the minimum degree to be numbered and eliminated first. At each following step i, a vertex of minimum degree in G_{i-1} is chosen as vertex i and eliminated, and ties are broken arbitrarily. This is clearly a greedy algorithm, with no guarantees on the quality of the resulting ordering. However, the orderings produced by minimum degree are surprisingly good with respect to fill in practice.

The time complexity of this approach is definitely $O(nm^+)$, since all degrees in G_{i-1} can be computed in $O(m^+)$ time at each step i. However, this requires $O(n + m^+)$ space, violating the $O(n + m)$ space requirement.

2.3 Supernodes

In a graph G, two adjacent vertices u and v are said to be *indistinguishable* if $adj(u) \cup \{u\} = adj(v) \cup \{v\}$. Clearly, if u and v are indistinguishable then they have the same degree, and if one of them, say u, is eliminated, no new fill edges joining v to any other neighbor of u are created. The degree of v will decrease by one (to reflect the elimination of u) in the remaining graph. Thus if one of them is among the vertices with minimum degree, then they both are, and after the elimination of one, the other will continue to be among the vertices with minimum degree in the next elimination graph. For this reason, both vertices could be eliminated at the same step, and numbered consecutively in a minimum degree ordering.

It is shown in [5] that two vertices that become indistinguishable at one step of the elimination game remain indistinguishable for the rest of the algorithm. In addition, they can be eliminated together whenever one of them is chosen for elimination [6]. Thus for purposes of the MD algorithm, the two vertices can be merged into a *supernode* and treated as one vertex for the remainder of the algorithm. This is called *mass elimination* in MD implementations.

At the beginning of the algorithm, all vertices are supernodes of size one. Then during the algorithm, indistinguishable supernodes are merged together as they are detected. It is common to use the *external degrees* of supernodes [7]: the external degree of a supernode is the number of vertices adjacent to it that belong to *other* supernodes. The *weight* of a supernode is the number of vertices that are absorbed in it.

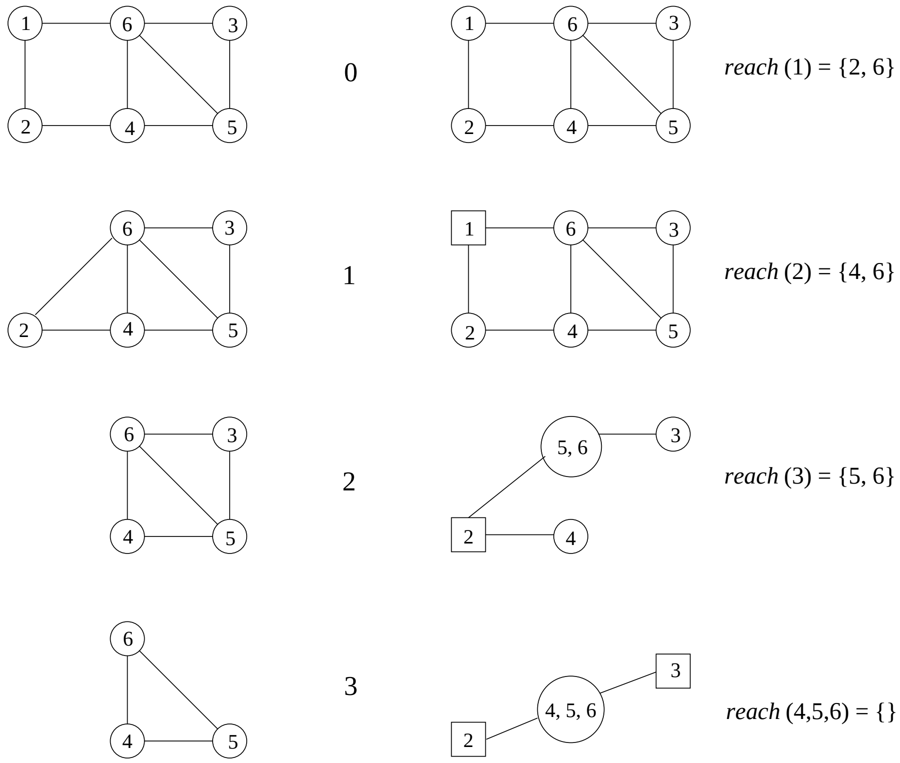

Figure 2: The elimination process illustrated with elimination graphs (column on the left) and quotient graphs (column on the right).

2.4 Quotient graph model

In the elimination graph model, the graph shrinks by one vertex at each step, but it might grow by many edges, and thus require significantly more space than the original graph. Quotient graphs [4] enable the ordering algorithm to use space bounded by the size of the original graph, and are used in all modern implementations of MD requiring $O(n+m)$ space.

The *quotient graph* $\mathcal{G}$ consists of two types of nodes: *snodes* and *enodes*. Initially, $\mathcal{G}_0$ is identical to the elimination graph G_0 and consists of only snodes (supernodes). When an snode is eliminated, it is not removed from the quotient graph, but it becomes an enode (eliminated supernode). In Figure 2, an example of the elimination is shown with both elimination graph and quotient graph representations. The snodes are drawn as circles, and enodes are drawn as squares. The adjacency set of an snode in the quotient graph is divided into its s-adjacency and its e-adjacency. The set of snodes adjacent to an snode r is denoted by $sadj(r)$, and the set of enodes adjacent to r is denoted by $eadj(r)$. Thus in the quotient graph, $adj(r) = sadj(r) \cup eadj(r)$.

The *reachable set* of an snode r, $reach(r)$, is the union of its s-adjacency and the snodes that it can reach through paths consisting of only enodes, and thus it corresponds to the neighbors in the elimination graph: $reach_{\mathcal{G}_i}(r) = adj_{G_i}(r)$. Consequently, to determine the next vertex to eliminate in MD, the sizes of the reachable sets of all candidate snodes must be computed. In order to make this more efficient, neighboring enodes are merged together so that a path consisting of only enodes is now shortened to one enode. Hence, $reach(r) = sadj(r) \cup (\cup_{e \in eadj(r)} sadj(e))$.

When an snode r is eliminated, r and all the enodes that are neighbors of r are merged into one enode. If r does not have any neighboring enodes then it becomes an enode by itself. The elimination of r could cause changes in the adjacency sets of other snodes as well. If two snodes become indistinguishable, they are merged together. If two adjacent snodes r and s have an enode e as a neighbor, then the edge joining r and s can be deleted from the quotient graph since it is redundant. (The snodes r and s are adjacent in the elimination graph since they are reachable from each other through e in the quotient graph.) This process is illustrated in Figure 2. The numbers in the middle indicate step k of the elimination process. The graphs on the left side represent the elimination graphs G_k, and the ones on the right side represent the quotient graphs $\mathcal{G}_k$ for each k.

3 Minimum Degree algorithms in detail

In the previous section, we introduced the idea of the minimum degree algorithm by considering the elimination of a single vertex in an elimination graph. However, practical implementations use the quotient graph data structure, and eliminate supernodes. In this section we present detailed algorithmic descriptions of several MD algorithms; all these are modern implementations based on quotient graphs and use the tools described in Section 2.4. Since we use the external degree of a supernode, the computed ordering might not in some cases correspond to a strict minimum degree ordering. However, the use of external degree tends to give better results than exact degree in practice [7].

3.1 Original Minimum Degree

The original MD algorithm, enhanced by the techniques mentioned in Section 2.4, is presented in Figure 3. We only discuss the details of the most time consuming steps.

Asymptotically, the costliest operation in MD is the degree update. After a vertex has been eliminated, the graph changes, and the degrees of the remaining nodes have to be recomputed in order to choose a vertex of minimum degree. Thinking in elimination graph terms, it is easy to see that only the neighbors of the eliminated vertex need to have their degrees recomputed. In the quotient graph, this corresponds to $reach_{\mathcal{G}_{k-1}}(u_k)$, where u_k is the supernode eliminated at step k. Thus we need to compute the reachable set of the snode to be eliminated. After the elimination, the snodes in the reachable set examine their own reachable sets to find their new degrees. These two steps correspond to the major steps in the MD algorithm described in Fig. 3.

We now study the time complexity of the MD algorithm given in Figure 3. Let n_p denote the total number of supernodes eliminated. At each step k, when snode u_k is to be eliminated in $\mathcal{G}_{k-1}$, the following steps are performed:

1. The enodes adjacent to u_k are merged into u_k.

2. The snodes adjacent to u_k and the snodes adjacent to the enodes merged with u_k are included in the reachable set. Note that each snode appears once in the reachable set since we mark the snodes when they are reached the first time. The computed reachable set is equal to $reach_{\mathcal{G}_{k-1}}(u_k)$.

```
G_0 = G;
Compute initial supernodes and their weights;
Compute initial degrees;
mark = 0; k = 0; t = 0;
while there are snodes in G_k do
    k = k + 1;
    choose u_k to be an snode of minimum degree;
    replace snode u_k with enode u_k;

    { 1. Find the reachable set of u_k }
    t = t + 1; reach = {};
    { 1a. Include snodes adjacent to u_k in reachable set }
    for each snode r ∈ sadj(u_k) do
        mark(r) = t; reach = reach ∪ r;
    {1b. Process enodes adjacent to u_k and include snodes adjacent to them in the reachable set }
    for each enode e ∈ eadj(u_k) do
        for each snode r ∈ sadj(e) with mark(r) < t do
            mark(r) = t; reach = reach ∪ r;
        merge e into u_k;

    Detect new supernodes;
    Form updated quotient graph G_k;

    {2. Update the degrees of snodes in the reachable set of u_k }
    for each snode r ∈ reach do
        t = t + 1; mark(r) = t; degree(r) = 0;
        {2a. Examine snodes adjacent to r }
        for each snode s ∈ sadj(r) do
            mark(s) = t; degree(r) = degree(r) + weight(s);
        {2b. Examine enodes adjacent to r and snodes adjacent to the enodes }
        for each enode e ∈ eadj(r) do
            for each snode s ∈ sadj(e) with mark(s) < t do
                mark(s) = t;
                degree(r) = degree(r) + weight(s);
n_p = k;
```

Figure 3: The MD algorithm.

3. For each snode r in the reachable set, we count each of its neighboring snodes s and each of its neighboring enodes e in $\mathcal{G}_{k-1}$ exactly once.

4. Finally, for each enode e that we reach in this fashion, the s-adjacency of e is also examined. This is done exactly once for each enode e in the e-adjacency of each snode r in the reachable set. However, in the worst case, the same enode e can belong to the e-adjacency of every snode r in the reachable set. Thus the adjacency of e might have to be examined once for every snode in the reachable set. This is illustrated in Figure 4.

As a consequence, the number of edges examined during a run of the algorithm is expressed as follows:

$$O\left(\sum_{k=1}^{n_p}\left(|sadj(u_k)| + \sum_{e\in eadj(u_k)} |sadj(e)| + \sum_{r\in reach(u_k)}\left(|sadj(r)| + \sum_{e\in eadj(r)} |sadj(e)|\right)\right)\right).$$

All sets appearing in this expression should have subscript $\mathcal{G}_{k-1}$ since we are considering adjacencies in this quotient graph.

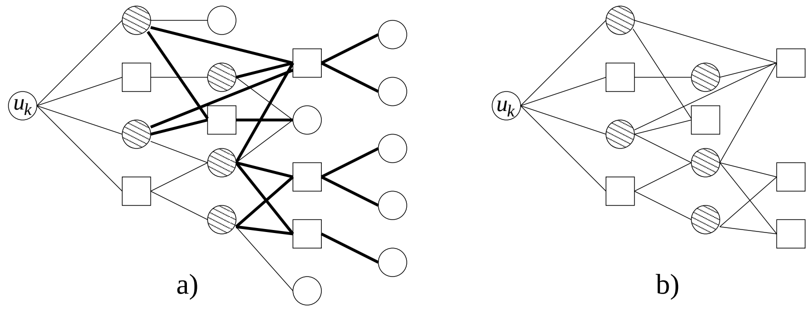

Figure 4: The local graphs searched by (a) the MD and MMD algorithms, and (b) the AMD algorithm. The node u_k is the current snode being eliminated; it becomes an enode in this step. The square nodes denote enodes, the hatched circles denote snodes in the reachable set of u_k, and the open circles denote additional snodes examined to update the degrees of the snodes in the reachable set. The thick lines represent edges that might be traversed several times at each step.

Theorem 1 *The running time of MD is $O(n^2m)$.*

Proof: Resolving the above sum term by term, the adjacencies of all the nodes in the graph is $O(m)$. The sum of the s-adjacencies of the enodes examined at a step is also $O(m)$. The *reach* set is bounded by $O(n)$; and the number of edges examined when considering the s-adjancencies of the *reach* sets is $O(m)$. Thus, we can see that the running time of MD is $O(n(m + (nm))) = O(n^2m)$. □

Depending on the graph and the snodes, n_p might be quite smaller than n, making the given theoretical bound too pessimistic. The graph needs also to be quite dense to meet the given bound, and as we get more and more cliques new supernodes will probably be formed, decreasing n_p. However, we will show at the end of this section that the given bound is tight by showing a simple graph that meets the given bound.

3.2 Multiple Minimum Degree

The Multiple Minimum Degree (MMD) algorithm, an improvement over the MD algorithm, was proposed by Liu [7]. Consider an independent set K of vertices. The elimination of a vertex in K cannot change the degree of any other vertex in this set, since no two vertices in K are adjacent. If we include only vertices of minimum degree in K, then clearly after the elimination of any vertex in K, the other vertices of K will still be among the minimum degree vertices at the next elimination step. The idea of the MMD algorithm is to eliminate a maximal independent set of minimum degree vertices before doing a degree update. At each step i of the algorithm, an independent set K_i of minimum degree vertices are found. These are eliminated and vertices adjacent to them are marked as vertices whose degrees need to be updated. The degrees of all the marked vertices are updated only after all the vertices in K_i are eliminated. In the quotient graph model, the set of snodes whose degrees need to be updated is the union of the reachability sets of the snodes in K_i. If these reachability sets have snodes in common, fewer degree updates would be needed than in the MD algorithm.

```
𝒢_0 = G;
Compute initial supernodes and their weights;
Compute initial degrees;
mark = 0; i = 0; t = 0;
while there are snodes in 𝒢_i do
    i = i + 1;
    choose K_i to be an independent set of snodes of minimum degree;
    t = t + 1; update = {};
    {1. Eliminate snodes in K_i;}
    { update is the union of the reachability sets of these snodes. }
    for each snode k ∈ K_i do
        mark(k) = ∞;
        for each snode r ∈ sadj(k) do
            mark(r) = t; update = update ∪ r;
        for each enode e ∈ eadj(k) do
            for each snode r ∈ sadj(e) with mark(r) < t do
                mark(r) = t; update = update ∪ r;
    {2. Update quotient graph after eliminating snodes in K_i.}
    for each snode k ∈ K_i do
        replace snode k with enode k;
        merge k and eadj(k) to one enode;
    Detect new supernodes;
    Form 𝒢_i;
    {3. Compute degrees of each snode in the update set.}
    for each snode r ∈ update do
        t = t + 1; mark(r) = t; degree(r) = 0;
        for each snode s ∈ sadj(r) do
            mark(s) = t; degree(r) = degree(r) + weight(s);
        for each enode e ∈ eadj(r) do
            for each snode s ∈ sadj(e) with mark(s) < t do
                mark(s) = t; degree(r) = degree(r) + weight(s);
n_h = i;
```

Figure 5: The MMD algorithm.

When the MMD idea is implemented with supernodes instead of single vertices, the assumed current minimum degree might become inaccurate. Since we use external degrees for snodes, eliminating an snode in the independent set K might actually cause an snode outside of K to acquire an external degree lower than that of any of the other snodes in K. In this case, eliminating the snodes of K before other snodes of possibly lower degree will generate a slightly perturbed minimum degree ordering. However, in practice the quality of the orderings from the MMD algorithm is usually even better than orderings from the MD algorithm with respect to fill.

If all the independent sets are of size one, then the work of MMD is equal to that of MD. The difference is that degree update is done less frequently when the independent sets are not just singletons. Let K_i be the set of independent supernodes that are eliminated at step i, and let n_h be the total number of steps. For each snode $k \in K_i$, we will do the same work as for each u_k in the MD algorithm to find $reach(u_k)$. However, the degree update is performed on all the snodes of $reach(K_i)$ at the same step i. Adding up the operations of the algorithm in a straight forward manner, we get:

$$O\left(\sum_{i=1}^{n_h}\left(\sum_{k\in K_i}|sadj(k)|+\sum_{e\in eadj(K_i)}|sadj(e)| + \sum_{r\in reach(K_i)}\left(|sadj(r)| + \sum_{e\in eadj(r)}|sadj(e)|\right)\right)\right).$$

Theorem 2 *The running time of MMD is $O(n^2m)$.*

Proof: The analysis is similar to the MD algorithm. At most $O(n)$ snodes can be in the total reachable set, and thus the time complexity of MMD is also $O(n(m + (nm))) = O(n^2m)$. □

For the MMD algorithm, the gap between n_h and n is even larger. Thus we can expect better performance of MMD than the given bound on average. However, the example at the end of this section shows that the given bound is tight.

3.3 Approximate Minimum Degree

Like MD, and unlike MMD, the Approximate Minimum Degree (AMD) algorithm is a single elimination algorithm; hence the degree and graph updates are performed after a single supernode is eliminated. The idea of the AMD algorithm is to compute an upper bound on the degrees inexpensively instead of computing the exact degrees, and to use this upper bound as an approximation to the degree for choosing supernodes to eliminate.

Let us define the *weight* of an snode to be the number of nodes in the original graph G_0 that are members of the supernode. We also define the weight of an enode e to be the sum of the weights of the snodes adjacent to it in the current quotient graph, i.e., $weight(e) = \sum_{s\in sadj(e)} weight(s)$. Let r be an snode whose degree is to be updated. The degree of r cannot be greater than the sum of the weights of all the snodes and the enodes adjacent to it in the current quotient graph. AMD starts with this upper bound as an approximation for the degree of r. However, the s-adjacency sets of the enodes in $eadj(r)$ might overlap, making the bound too loose, and causing a large gap between the real degree and the approximated degree bound of r. This gap can be reduced by computing a quantity $diff(e)$ associated with each enode [1, 2] to remove some of the overlap in the adjacency sets.

Let u_k be the snode that is eliminated at step k. It is then merged with all of its e-neighbors, and the weight of the new giant enode u_k in the quotient graph $\mathcal{G}_k$ becomes the sum of the weights of all the snodes $r \in reach_{\mathcal{G}_{k-1}}(u_k)$:

$$weight(u_k) = \sum_{r\in reach_{\mathcal{G}_{k-1}}(u_k)} weight(r).$$

Since each snode r in the reachability set above is a neighbor of enode u_k in $\mathcal{G}_k$, the value $weight(u_k)$ will be added to the approximate degree of r. Therefore, for all the other enodes $e \in eadj(r)$ where $e \neq u_k$, to prevent double counting, we should include in $weight(e)$ only the contribution from the weights of the snodes disjoint from those in the reachability set; i.e, we should sum only the weights of snodes $s \in sadj(e) \setminus reach_{\mathcal{G}_{k-1}}(u_k)$ instead of summing the weights of all snodes in $sadj(e)$.

We define a *diff* function for enodes $e \in eadj(reach_{\mathcal{G}_{k-1}}(u_k))$ in the quotient graph $\mathcal{G}_k$ as

$$diff(e) = \begin{cases} weight(e) & \text{if } e = u_k, \\ weight(e) - \sum_{r\in(reach_{\mathcal{G}_{k-1}}(u_k)\cap sadj(e))} weight(r) & \text{if } e \neq u_k. \end{cases}$$

The approximate degree of $r \in reach_{\mathcal{G}_{k-1}}(u_k)$ can be then computed from:

$$adegree(r) = weight(u_k) + \sum_{s \in sadj(r)} weight(s) + \sum_{e \in eadj(reach(u_k))} diff(e).$$

The AMD algorithm is described in Figure 6. The local graph that is searched is shown in Fig. 4. Note that now each edge in this local graph is examined at most twice, once from each of its endpoints.

$\mathcal{G}_0 = G$;
Compute initial supernodes and their weights;
for each snode $r \in \mathcal{G}_0$ **do**
    $sdegree(r) = 0$;
    for each snode $s \in sadj(r)$ **do**
        $sdegree(r) = sdegree(r) + weight(s)$;
$mark = 0$; $k = 0$; $t = 0$;
while there are snodes in $\mathcal{G}_k$ **do**
    $k = k + 1$;
    {1. Eliminate an snode u_k and compute its reachable set.}
    choose u_k to be an snode of minimum approximate degree;
    $kweight = weight(u_k)$;
    replace snode u_k with enode u_k;
    $t = t + 1$; $reach = \{\}$; $weight(u_k) = 0$;
    {1a. Include snodes adjacent to u_k in the reachable set}
    for each snode $r \in sadj(u_k)$ **do**
        $mark(r) = t$; $reach = reach \cup r$;
        $weight(u_k) = weight(u_k) + weight(r)$;
        $sdegree(r) = sdegree(r) - kweight$;
    {1b. Include snodes that are neighbors of enodes adjacent to u_k in the reachable set }
    for each enode $e \in eadj(u_k)$ **do**
        for each snode $r \in sadj(e)$ with $mark(r) < t$ **do**
            $mark(r) = t$; $reach = reach \cup r$;
            $weight(u_k) = weight(u_k) + weight(r)$;
        let u_k absorb e;
    Detect new supernodes; Form $\mathcal{G}_k$; $t = t + 1$;
    {2a. Compute *diff*(e) for enodes adjacent to snodes in the reachability set.}
    for each snode $r \in reach$ **do**
        for each enode $e \in eadj(r)$, $e \neq u_k$ **do**
            if $mark(e) < t$ **then**
                $diff(e) = weight(e) - weight(r)$; $mark(e) = t$;
            else
                $diff(e) = diff(e) - weight(r)$;
    {2b. Compute approximate degrees for snodes in the reachability set.}
    for each snode $r \in reach$ **do**
        $adegree(r) = sdegree(r) + weight(u_k) - weight(r)$;
        for each enode $e \in eadj(r)$, $e \neq u_k$ **do**
            $adegree(r) = adegree(r) + diff(e)$;
$n_p = k$;

Figure 6: The AMD algorithm.

Because of the increased difficulty of finding the set intersections, multiple elimination is usually not implemented in AMD. Without the multiple elimination, the total number of steps in the AMD algorithm is:

$$O\left(\sum_{k=1}^{n_p}\left(|adj(u_k)| + \sum_{e \in eadj(u_k)} |sadj(e)| + \sum_{r \in reach(u_k)} |eadj(r)|\right)\right).$$

Theorem 3 *The running time of AMD is $O(nm)$.*

Proof: In the expression above, the second and the third terms together is $O(m)$. Hence the complexity is $O(n(m+m)) = O(nm)$. $\square$

For AMD, Amestoy, Davis, and Duff [1] have shown a tighter time complexity of $O(m^+)$ on bounded degree graphs, still with quotient graphs and $O(n+m)$ space bound.

3.4 Examples that meet the bounds

Consider the following graph on $8k+1$ vertices (shown in Figure 7 for $k=1$): There are $4k$ "outer" vertices $x_1, \ldots, x_{4k}$, $4k$ "inner" vertices $y_1, \ldots, y_{4k}$, a "hub" vertex z, an edge between each x_i and each y_j with $|i-j| \neq 2k$, and an edge between each y_j and z.

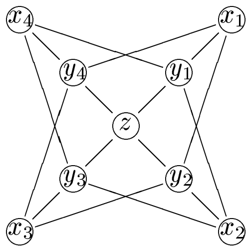

Figure 7: An example on which MD requires $\Theta(n^2 m)$ time.

Clearly MD eliminates the $4k$ outer vertices first and, with the right tie-breaking strategy, does so in the order $x_1, \ldots, x_{4k}$. At the time that each of the k outer vertices $x_{k+1}, \ldots, x_{2k}$ is eliminated, it is distinguishable and adjacent to at least k distinguishable inner vertices (including $y_1, \ldots, y_k$). Each of these inner vertices is adjacent to at least k unmerged enodes (including $x_1, \ldots, x_k$), and each of these enodes is adjacent to at least k distinguishable inner vertices (including $y_1, \ldots, y_k$). Thus the total work to update degrees while eliminating these outer vertices is $\Omega(k^4)$. Consequently, MD requires $\Theta(n^2 m)$ time on this example since $n = 8k+1$ and $e = 4k^2$. By the same arguments, AMD requires $\Theta(k^3) = \Theta(nm)$ time on the same example.

An example for MMD is slightly more complicated. Beginning with the graph above, add a clique with $4k$ vertices $c_1, \ldots, c_{4k}$, add edges between x_i and $c_1, \ldots, c_{i-1}$ for each i, add edges between each y_j and each c_ℓ, and add edges between z and each c_ℓ. Then MMD first eliminates the outer vertices *one at a time* in the same order as above, so the work is again $\Omega(k^4)$, resulting in $\Theta(n^2 m)$ time.

4 Conclusions

We have given a thorough analysis of the MD algorithm together with its variants MMD and AMD. Based on quotient graph implementations and $O(n+m)$ space requirement, we have established an $O(n^2 m)$ time bound for MD and MMD, and an $O(nm)$ bound for AMD. Note that these bounds are for nearly dense graphs. Fortunately, these bounds are not often observed for problems that are solved in practice.

A further development of this work is to identify graph classes with provably better MD time complexities.

Acknowledgments. We thank Prof. Tim Davis of the University of Florida for his helpful comments.

References

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., 18 (1996), pp. 140–158.

[3] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.

[4] J. A. George and J. W. H. Liu, *A quotient graph model for symmetric factorization*, in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., SIAM Publications, 1978, pp. 154–175.

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

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

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

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

[9] S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, 3 (1961), pp. 119–130.

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

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

[12] R. E. Tarjan, *Graph theory and Gaussian elimination*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 3 – 22.

[13] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55 (1967), pp. 1801–1809.

[14] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 77–79.