Science & Technology Facilities Council

Technical Report

# A note on fast approximate minimum degree orderings for symmetric matrices with some dense rows

H. S. Dollar J. A. Scott

March 7, 2008

RAL-TR-2007-022

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk



**ISSN 1358-6254**

**RAL-TR-2007-022**

# A note on fast approximate minimum degree orderings for symmetric matrices with some dense rows

H. S. Dollar[1], J. A. Scott[1]

## ABSTRACT

Recently a number of variants of the approximate minimum degree algorithm have been proposed that aim to efficiently order symmetric matrices containing some dense rows. We compare the performance of these variants on a range of problems and highlight their limitations. This leads us to propose a new variant that offers both speed and robustness.

**Keywords:** approximate minimum degree ordering algorithm, sparse symmetric matrices, dense rows, graph algorithms, ordering algorithms.

**AMS(MOS) subject classifications:** 65F50, 65F05.

---

[1] Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England, UK.
Email: s.dollar@rl.ac.uk & j.a.scott@rl.ac.uk
Current reports available from `http://www.numerical.rl.ac.uk/reports/reports.html`
This work was supported by EPSRC grant EP/E053351/1.

March 7, 2008

# 1 Introduction

The efficiency of sparse direct solvers for the solution of symmetric linear systems $Ax = b$, in terms of both the storage needed and work performed, is dependent upon the order in which the variables are eliminated, that is, the order in which the pivots are selected. Many solvers include a preordering step that aims to use information on the sparsity pattern of $A$ to find a permutation so that, if the pivots are chosen in order from the diagonal of the permuted matrix, the computed factors are sparser than if the pivots were chosen in order from the diagonal of the original matrix. If $A$ is positive definite, the pivot sequence chosen from the sparsity pattern alone can be used by the factorization phase without modification and a Cholesky factorization $PAP^T = LL^T$, where $P$ is a permutation matrix and $L$ is lower triangular, can be computed. More generally, numerical pivoting must be incorporated during the factorization phase to maintain numerical stability and, in this case, the pivot sequence computed by the symbolic analyse phase may have to modified.

The problem of finding a permutation $P$ that results in the smallest amount of fill-in for a Cholesky factorization is NP-complete [15] and so heuristics are used to find a good ordering. Two main classes of methods are widely used: those based on nested dissection [9] and those based on the minimum degree algorithm [14]. In recent years, nested dissection has often been found to be the method of choice for many very large problems (typically those of order greater than 50,000) [8]. However, it can be more expensive than the most efficient implementations of the minimum degree algorithm, which is preferred for more modest size problems and for very sparse problems. Currently, the most successful variant of the minimum degree algorithm is the approximate minimum degree algorithm (AMD) and, in particular, the AMD algorithm introduced by Amestoy, Davis and Duff [1] is widely used. The AMD algorithm is more efficient since it uses computationally cheap bounds on the minimum degree in place of the exact minimum degree and, in practice, it produces orderings that are comparable in quality [1].

Although AMD is generally very successful, an important exception is when the matrix $A$ has some dense (or almost dense) rows and columns. In this case, the run time for AMD can be high. AMD uses the undirected graph of the matrix and selects each node in turn to have minimum (approximate) degree. Once a node is selected, it is eliminated from the graph and replaced by adding edges between its neighbours so that the neighbours become a clique. If a row is full, the corresponding node will always be adjacent to the eliminated node so its adjacency list has to be scanned and updated, requiring $\mathcal{O}(n^2)$ operations for a problems with $n$ variables. This makes the algorithm prohibitively expensive.

We will compare three variants of the AMD algorithm that aim to efficiently form a pivot order when the matrix $A$ has some dense rows and columns. The first two were proposed by Carmen [3] and Davis [5]: each performs a preprocessing stage during which the rows and columns that it considers to be dense are removed, and then the AMD algorithm is applied to the remaining matrix. The variant that was originally introduced by Amestoy for use within the parallel direct solver `MUMPS` [11] and was recently discussed by Amestoy *et al.* [2] uses a more sophisticated dynamic partition of the rows into dense and sparse rows to efficiently compute the pivot order. Our experiments have shown that all three variants can perform poorly (either in terms of the quality of the pivot order or the time taken to compute it) and this leads us to present a new variant that combines the speed of the methods of Carmen and Davis with the robustness of the Amestoy method.

This paper is organised as follows. In Section 2, we review the minimum degree and AMD algorithms. The methods of Carmen, Davis, and Amestoy *et al.* are described and compared in Section 3. The new variant of the AMD algorithm that we propose is introduced in Section 4. In Section 5, we draw our conclusions.

# 2 The minimum degree and AMD algorithms

The minimum degree and AMD algorithms may be presented using the graph model of Rose [12, 13]. The nonzero pattern of a sparse symmetric matrix $A$ of order $n$ can be represented by an undirected graph

$G^0 = \left(V^0, E^0\right)$ with nodes $V^0 = \{1, \ldots, n\}$ and edges $E^0$. An edge $(i, j)$ is present in $E^0$ if and only if $a_{ij} \neq 0$ and $i \neq j$. Nodes $i$ and $j$ are adjacent to each other (neighbours) in graph $G^0$ if the edge $(i, j)$ is present in $E^0$.

The elimination graph $G^k = \left(V^k, E^k\right)$ describes the nonzero pattern of the *reduced matrix* $A^{(k)}$ of order $n^{(k)}$ still to be factored after $k$ pivots have been chosen and eliminated. The external degree $d_i^k$ of node $i$ is defined to be the number of nodes adjacent to node $i$ in $G^k$. The minimum degree algorithm chooses a node $p$ that has minimal external degree in the graph $G^{k-1}$ as the $k$th pivot. Algorithm 1 gives an outline of the minimum degree algorithm. The term *variable* is used to indicate a node that has not been removed from the elimination graph.

**Algorithm 1** The minimum degree algorithm

For each variable $i \in V^0$, set $d_i^0$ to be the external degree of $i$
$k = 1$
**while** $k \leq n$ **do**
  Select variable $p \in V^{k-1}$ to minimize $d_p^{k-1}$
  Eliminate $p$
  For each variable $i$ adjacent to $p$ in $G^{k-1}$, update $d_i^k$
  $k = k + 1$
**end while**

The graph $G^k$ depends on $G^{k-1}$ and the choice of the $k$th pivot. $G^k$ is found by selecting the $k$th pivot from $V^{k-1}$, adding edges to $E^{k-1}$ to make the nodes adjacent to $p$ in $G^{k-1}$ a *clique* (a fully connected subgraph) and then removing $p$ (and its edges) from the graph. The edges added to the graph correspond to fill-in. This addition of edges means that, if $G^k$ is stored explicitly, we cannot know the storage requirements in advance. To remedy this a quotient graph is used instead of an elimination graph [1]. However, the calculation of the external degree of a variable may then be expensive. Alternatively, the AMD algorithm calculates an upper bound, $\overline{d}_i$, on the external degree of a variable and then uses the upper bounds when choosing the next pivot. This generally results in the AMD algorithm being more efficient. Numerical results have shown that the AMD algorithm also produces orderings that are comparable in quality to the best classical minimum degree algorithms [1].

Unfortunately, the AMD algorithm can be very inefficient when the matrix $A$ has some dense (or almost dense) rows and columns. When a pivot $p$ is eliminated all of its adjacent neighbours must update their upper bounds $\overline{d}_i$. If $i$ is (almost) dense, it is (almost) certainly adjacent to $p$ and updating $\overline{d}_i$ will involve a large number of comparisons. Thus (almost) every step is expensive. For clarity, throughout the remainder of this paper, we will refer to this variant that takes no account of dense rows and columns as the *classical AMD algorithm*.

In Table 1, we list our test set. Each example is available from the University of Florida Sparse Matrix Collection [4] and, if unsymmetric, has been symmetrized by working with the sum of the given matrix and its transpose. The order $n$, the number of off-diagonal nonzero entries $nz$, the maximum external degree $d_{max}$, the average external degree $\mu$, and the standard deviation $\sigma$ of the external degrees of each (symmetrized) matrix is given. The listed matrices include those that Amestoy *et al.* found to contain dense (or near dense) rows and have $\sigma/\mu \geq 1$. Amestoy *et al.* found that, for matrices with $\sigma/\mu < 1$, classical AMD performed well and so we restrict attention to those with $\sigma/\mu \geq 1$.

We initially compare the performance of the classical minimum degree algorithm with the classical AMD algorithm. The versions of these algorithms that are in the `HSL` [10] sparse direct solver `MA57` [6] (version 3.2.0) are used. In Table 2, we give the time to obtain the ordering and the forecast number of reals required to hold the matrix factor. Throughout this paper, the latter piece of information is used as a measure for the quality of the ordering and is generated by passing the pivot order to the analyse phase of `MA57`. All numerical results were obtained using a 3.6 GHz Intel Xeon dual processor Dell Precision

| Problem | $n$ | $nz$ | $d_{max}$ | $\mu$ | $\sigma$ | $\sigma/\mu$ |
|---|---|---|---|---|---|---|
| lpl1 | 32460 | 328036 | 253 | 10.1 | 15.5 | 1.53 |
| gupta3 | 16783 | 9323427 | 14672 | 556 | 1234 | 2.22 |
| mip1 | 66463 | 10352819 | 66395 | 156 | 351 | 2.25 |
| net-4 | 88343 | 2441727 | 4791 | 27.6 | 85.3 | 3.09 |
| ckt11752_dc_1 | 49702 | 327834 | 2921 | 6.60 | 24.6 | 3.73 |
| rajat23 | 110355 | 559733 | 3336 | 5.07 | 19.3 | 3.80 |
| rajat22 | 39899 | 198081 | 3336 | 4.96 | 24.8 | 4.99 |
| gupta2 | 62064 | 4248286 | 8413 | 68.5 | 356 | 5.20 |
| gupta1 | 31802 | 2164210 | 8413 | 68.1 | 360 | 5.29 |
| a0nsdsil | 80016 | 355034 | 5003 | 4.44 | 50.0 | 11.3 |
| blockqp1 | 60012 | 640033 | 40011 | 10.7 | 305 | 28.6 |
| trans5 | 116835 | 827879 | 114191 | 7.09 | 371 | 52.3 |
| ins2 | 309412 | 2751484 | 309412 | 8.89 | 590 | 66.4 |
| rajat29 | 643994 | 4026512 | 454043 | 6.25 | 783 | 125 |

Table 1: The test set: order $n$, number of nonzeros $nz$, the maximum external degree $d_{max}$, the mean $\mu$ of the external degrees, and their standard deviation $\sigma$.

| | Minimum Degree | | Classical AMD | |
|---|---|---|---|---|
| Problem | Time | $nz(L)$ | Time | $nz(L)$ |
| lpl1 | 1.22 | 0.97 | 0.28 | 0.97 |
| gupta3 | 115 | 5.72 | 8.74 | 5.72 |
| mip1 | 40.8 | 38.9 | 10.2 | 39.0 |
| net4-1 | 9.28 | 2.46 | 1.79 | 2.38 |
| ckt11752_dc_1 | 0.49 | 0.59 | 0.19 | 0.56 |
| rajat23 | 0.65 | 0.45 | 0.33 | 0.46 |
| rajat22 | 0.29 | 0.15 | 0.13 | 0.15 |
| gupta2 | 2020 | 5.86 | 88.9 | 5.89 |
| gupta1 | 405 | 2.02 | 28.9 | 2.06 |
| a0nsdsil | 14.6 | 0.34 | 2.70 | 0.34 |
| blockqp1 | 47.4 | 0.38 | 8.11 | 0.38 |
| trans5 | 246 | 0.68 | 123 | 0.68 |
| ins2 | 2080 | 1.53 | 825 | 1.53 |
| rajat29 | $> 10^4$ | — | 4040 | 9.77 |

Table 2: Comparison of the times and the predicted number of reals in $L$ ($\times 10^6$) generated by orderings from the minimum degree algorithm and the classical approximate minimum degree algorithm.

670 with 4 Gbytes of RAM that was running Red Hat Enterprise Linux Server release 5.1 (kernel 2.6.18-53.1.13.el5). The Nag f95 compiler with the -O4 option was used. All times are CPU times in seconds. For `rajat29`, the minimum degree algorithm failed to compute a pivot order within 10000 seconds and, hence, no results are given.

As expected, we observe that the classical AMD algorithm is significantly more efficient than the minimum degree algorithm for our test problems while the quality of the orderings is comparable. Observe that the times to obtain the elimination orderings differ by at least an order of magnitude for the `gupta*` problems. However, as Amestoy *et al.* [2] show, for problems in our test set, the times to generate the elimination ordering with the classical AMD algorithm are very high when compared with problems of similar size and number of entries but no dense rows: the algorithms described in Section 3 aim to substantially reduce these times while maintaining the quality of the ordering.

# 3 Variants of AMD for efficiently detecting and treating dense rows

The problem of forming variants of the AMD algorithm that efficiently detect and treat dense rows has recently been considered by Carmen [3], Davis [5] and Amestoy *et al.* [2]. Using the names given by their authors, we refer to these variants as AMDpre, CS_AMD and QAMD, respectively. The `amd` function of MATLAB® (version 7.5) uses the CS_AMD algorithm.

## 3.1 The AMDpre and CS_AMD algorithms

The AMDpre [3] and CS_AMD [5] algorithms both use a preprocessing step to search the matrix $A$ for rows that they consider to be dense. The matrix $A$ is reordered to take the form

$$A = \left[\begin{array}{c|c} A_1 & A_r^T \\ \hline A_r & A_d \end{array}\right],$$

where $A_r$ and $A_d$ are considered dense. The classical AMD algorithm is then applied to $A_1$ and the dense rows are appended to the end of the resulting pivot ordering: the authors assume that the dense rows all experience roughly the same amount of fill-in and order them in increasing value of their external degree within $A$.

The CS_AMD algorithm classifies a row in $A$ as dense if its external degree is greater than $\alpha\sqrt{n}$, where $\alpha > 0$ is a fixed constant (the default is $\alpha = 10$). We remark that Reid incorporated an option that uses a similar idea into the implementation of the minimum degree algorithm within the `HSL` [10] sparse solver `MA27` [7].

The AMDpre algorithm uses a procedure that is equivalent to that given in Algorithm 2. The threshold $\gamma$ takes the form $\gamma = \beta\sqrt{n}$, where $\beta > 0$ (the default value is $\beta = 1$). The major difference between this and the simpler CS_AMD variant is that AMDpre updates the external degrees when a row is selected as dense and removed from $A_1$. This results in a more complicated implementation but a smaller threshold can be used to remove roughly the same number of dense rows.

## 3.2 The QAMD algorithm

The QAMD algorithm was developed independently of the AMDpre and CS_AMD algorithms. It was originally used by Amestoy in the parallel direct solver `MUMPS`[11]. It uses a somewhat different approach since the partitioning of the matrix is dynamic, allowing the matrix to be partitioned more than once as the ordering proceeds. In [2], Amestoy *et al.* define a row to be full if all of its entries are (symbolically)

**Algorithm 2** The AMDpre method for choosing $A_1$

Set $A_1 = A$
Calculate the external degrees of the rows in $A_1$
**while** maximum external degree $\geq \gamma$ **do**
  Remove from $A_1$ the row (and corresponding column) that has largest external degree
  Update the external degrees of the rows in $A_1$
**end while**

nonzero; a row is quasi dense if it has a high proportion of nonzero entries (so that its external degree is large); a row is sparse if it is neither full nor quasi dense. Amestoy *et al.* partition the matrix $A$ as

$$A = \left[\begin{array}{c|c|c} A_s & A_{q1}^T & A_{f1}^T \\ \hline A_{q1} & A_q & A_{f2}^T \\ \hline A_{f1} & A_{f2} & A_f \end{array}\right],$$

where the rows in $A_s$ are sparse, the rows in $A_q$ are quasi dense, and the rows in $A_f$ are full. QAMD applies the AMD algorithm to the submatrix $A_s$ but, if a row becomes quasi dense or full, its variable is removed from $A_s$ and placed into $A_q$ or $A_f$, respectively. When all of the variables in $A_s$ have been either eliminated or reclassified, the exact external degrees of the quasi dense rows are calculated and these rows are then reclassified to be either sparse or full and the algorithm restarts. The algorithm can restart a number of times until only full variables remain uneliminated. The corresponding variables are appended to the end of the pivot order (the order in which they are appended is arbitrary).

A threshold $\tau > 0$ is used to select quasi dense rows. In [2], a variable $i$ is reclassified as quasi dense if is not known to be full and $\overline{d}_i^s + n_q + n_f \geq \tau$, where $n_q$ and $n_f$ are the numbers of quasi dense and dense rows, respectively, and $\overline{d}_i^s$ is the approximate degree of variable $i$ with respect to the matrix $A_s$. The threshold is updated at the beginning of each restart using a definition of the form $\tau = \tau(\mu, \sigma)$. A more detailed description of the QAMD algorithm and the choice of $\tau$ is given in [2].

We remark that, although the QAMD algorithm can be implemented using the same amount of memory as the classical AMD algorithm, the implementation is complicated because the quotient graph needs to be reformed each time during each restart.

### 3.3 Comparison of AMDpre and QAMD

In Table 3, we compare the performance of the AMDpre, CS_AMD and QAMD methods with that of the classical AMD algorithm. The default parameters for each method are used. We have highlighted the best times (and those within 10 percent of the best) and the best values of $nz(L)$ for each problem. We observe that, as expected, there is generally a significant reduction in the time required to compute the ordering when the problem has some rows that are dense. AMDpre is generally faster than both CS_AMD and QAMD. For some problems, CS_AMD fails to detect many of the dense rows and this results in a slow run time. For example, for `gupta2`, CS_AMD finds 258 dense rows while AMDpre finds 1193, leading to the CS_AMD time being almost 40 times slower than AMDpre. Since both variants produce orderings of comparable quality, we favour AMDpre and will only consider the AMDpre and QAMD algorithms in the remainder of this section.

For some problems, for example `gupta*`, AMDpre is at least an order of magnitude faster than QAMD. This is because AMDpre only needs to store the structure of the submatrix $A_1$ but QAMD requires the whole structure to be stored and occasionally searched to allow the method to restart correctly. In terms of the quality of the ordering, for some problems (including `lpl1` and `net4-1`), AMDpre outperforms QAMD but for others (including `mip1` and `a0nsdsil`) the converse is true.

| Problem | | classical | AMDpre | CS_AMD | QAMD |
|---|---|---|---|---|---|
| lpl1 | Time | 0.28 | 0.27 | 0.28 | **0.19** |
| | $nz(L)$ | *0.97* | 0.99 | *0.97* | 1.20 |
| gupta3 | Time | 8.74 | **0.14** | 0.55 | 4.30 |
| | $nz(L)$ | 5.72 | 5.60 | *5.35* | 6.52 |
| mip1 | Time | 10.2 | **0.40** | 0.96 | 0.81 |
| | $nz(L)$ | *39.0* | 44.8 | 39.3 | 39.7 |
| net4-1 | Time | 1.79 | **0.49** | 0.58 | 0.78 |
| | $nz(L)$ | 2.38 | 2.39 | *2.37* | 3.09 |
| ckt11752_dc_1 | Time | 0.19 | **0.11** | 0.15 | 0.15 |
| | $nz(L)$ | *0.56* | *0.57* | *0.56* | 0.63 |
| rajat23 | Time | 0.33 | **0.19** | 0.29 | 0.23 |
| | $nz(L)$ | *0.46* | *0.47* | *0.46* | *0.47* |
| rajat22 | Time | 0.13 | **0.05** | **0.06** | **0.06** |
| | $nz(L)$ | *0.15* | *0.15* | *0.15* | *0.16* |
| gupta2 | Time | 88.9 | **0.37** | 13.4 | 5.74 |
| | $nz(L)$ | 5.89 | 6.30 | *5.77* | 5.98 |
| gupta1 | Time | 28.9 | **0.09** | 2.98 | 2.84 |
| | $nz(L)$ | 2.06 | 2.08 | *2.00* | 2.06 |
| a0nsdsil | Time | 2.70 | **0.07** | **0.07** | 0.09 |
| | $nz(L)$ | *0.34* | 0.39 | 0.39 | 0.35 |
| blockqp1 | Time | 8.11 | **0.03** | **0.03** | **0.04** |
| | $nz(L)$ | *0.38* | *0.38* | *0.38* | *0.38* |
| trans5 | Time | 123 | **0.15** | **0.16** | 0.22 |
| | $nz(L)$ | *0.68* | *0.69* | *0.69* | *0.70* |
| ins2 | Time | 825 | **0.31** | **0.30** | 0.53 |
| | $nz(L)$ | *1.53* | 1.59 | 1.59 | 1.59 |
| rajat29 | Time | 4040 | **2.34** | 2.62 | 3.32 |
| | $nz(L)$ | *9.77* | 9.97 | 9.87 | 11.7 |

Table 3: Comparison of the times and the predicted number of reals in $L$ ($\times 10^6$) generated by orderings from the classical AMD, AMDpre, CS_AMD and QAMD algorithms.

A potential problem with AMDpre is that it can remove a large number of sparse rows and this will result in the quality of the ordering being lost. To illustrate this, we used the HSL package YM11 to generate a banded random matrix with $n = 60000$, bandwidth equal to $2\sqrt{n}$, $nz = 20436000$, $d_{max} = 405$ and $\mu = 340$. The classical AMD and QAMD methods both calculate an ordering in 1.78 seconds with $nz(L) = 5.36 \times 10^8$. By comparison, AMDpre removes a total of 18661 rows/columns and takes 1.26 seconds. However, the quality of the pivot order has been lost as $nz(L) = 4.83 \times 10^9$.

Conversely, we can produce test problems where the AMDpre method will struggle to detect the dense rows. Suppose $A$ is of the form

$$A = \begin{bmatrix} S & U^T \\ U & T \end{bmatrix}, \tag{1}$$

where $S$ is a tridiagonal matrix, only the top right hand entry of $U$ is nonzero, and $T$ is a matrix from our test set that contains some dense rows. No matter how large the order of $S$ becomes, it will be the same rows of $T$ that cause the slow down in the efficiency of AMD and, hence, roughly the same number of rows should be removed for different dimensions of $S$. If we apply AMDpre to $A$, then as the order of $S$ increases, the threshold $\gamma = \beta\sqrt{n}$ also increases. Therefore, rows that were classified as dense for smaller orders may no longer be classified as dense. Suppose we set $T$ to be the matrix gupta1 and let the order of $A$ be $mn$ ($m = 1, 2, \ldots$). In Figure 1, we compare the time to calculate the elimination ordering of the matrix with different values of $m$. As $m$ increases, the time increases at a faster rate for AMDpre than QAMD. In Figure 2, we plot the number of rows that are detected as dense and, hence, removed during the preprocessing stage of AMDpre. As expected, as $m$ increases, the number of rows detected as dense drops substantially. Both variants produce orderings of comparable quality. Note that CS_AMD would suffer in a similar manner as the problem size increases. The behaviour seen in Figure 2 was observed when $T$ was chosen to be any of the problems in our test set but for some problems, the time difference was less marked than we see in Figure 1.

More generally, Algorithm 2 will fail to correctly detect all of the dense rows in a matrix if the maximum degree is smaller than $\beta\sqrt{n}$ but still significantly larger than the average external degree of the resulting matrix $A_1$. This suggests that we require a method that combines the power of the QAMD method for detecting dense rows with the efficiency of the AMDpre method.

# 4 The AMDD algorithm

The results in the previous section have highlighted the weaknesses of the existing attempts to modify the AMD algorithm to improve performance in the case of some dense rows. In this section, we introduce a new variant that is designed to overcome these weaknesses.

Let $i$ be a dense row. If row and column $i$ are removed from the matrix $A$, we expect the mean of value of the degrees of the variables to be reduced significantly more than if $i$ were a sparse row. This suggests that, instead of comparing the degree of a variable against the threshold $\gamma = \beta\sqrt{n}$ as used in the preprocessing stage of the AMDpre method, we should compare the mean before and after row/column $i$ is removed. Consider again the matrix $A$ given by (1). As the order of $A$ is increased, the difference in the value of the mean external degree before and after a row (and corresponding column) is removed decreases. Numerical experimentation on the problems in our test set and the additional problems considered in Section 3.3 has shown that defining $t(n_1)$ as

$$t(n_1) = \frac{\delta \ln(n_1)}{n_1},$$

where $\delta = 40$, produces a variant of AMD that is both efficient and maintains the quality of the ordering. the method was not found to be very sensitive to the choice of $\delta$ : values between 30 and 50 produced, in general, similar results.

Note that, if $\mu_1$ is the mean external degree of $A_1$, removing row and column $i$ from $A_1$ results in a

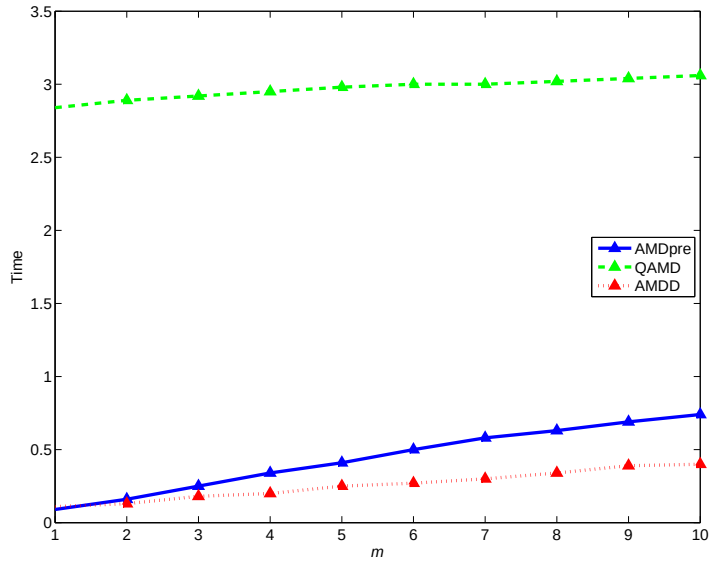


Figure 1: Time to calculate the ordering for the matrix $A$ defined by (1), where $T$ is set to the matrix `gupta1` and the dimension of $A$ is $nm$ ($m = 1, 2, \ldots$) using the AMDpre, QAMD and AMDD methods.

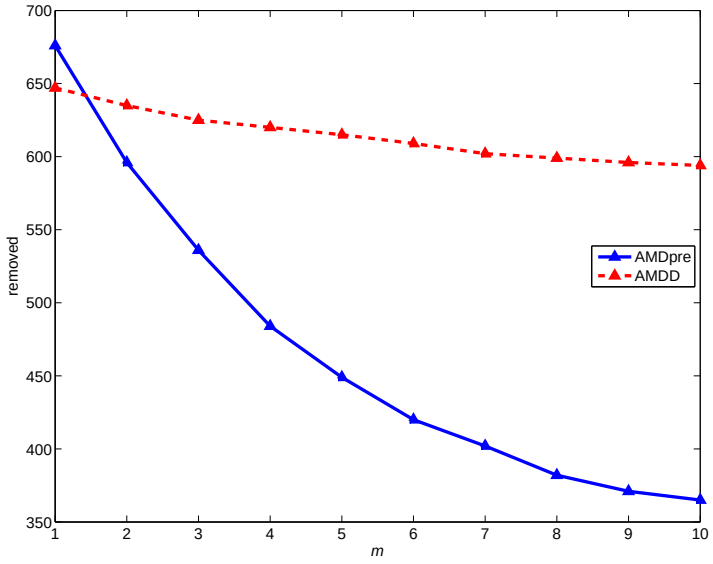


Figure 2: Number of dense rows detected and removed by the AMDpre and AMDD methods for the matrix $A$ defined by (1), where $T$ is set to the matrix `gupta1` and the dimension of $A$ is $mn$ ($m = 1, 2, \ldots$).

**Algorithm 3** The AMDD method for choosing $A_1$

Set $A_1 = A$, $n_1 = n$ and $q$ to be an empty list
For each row $i \in A_1$, set $d_i$ to be the external degree of $i$
Calculate the mean $\mu_1$ of the external degrees of the rows in $A_1$
Select row $i$ from $A_1$ that has largest external degree
**while** $\mu_1 - \frac{\mu_1 n_1 - 2d_i}{n_1 - 1} \geq t(n_1)$ **do**
  Add $i$ to the beginning of the list $q$
  Remove row and column $i$ from $A_1$
  $n_1 = n_1 - 1$
  Update $t(n_1)$
  Update the mean $\mu_1$ of the external degrees of the rows in $A_1$
  Select row $i$ from $A_1$ that has largest external degree
**end while**
Apply classical AMD to $A_1$ to form a pivot order
Append the list $q$ to the end of this pivot order to form a pivot order for $A$

| | QAMD | | AMDD | |
|---|---|---|---|---|
| Problem | Time | $nz(L)$ | Time | $nz(L)$ |
| lpl1 | **0.19** | 1.20 | 0.28 | *0.99* |
| gupta3 | 4.30 | 6.52 | **0.17** | *5.54* |
| mip1 | 0.81 | *39.7* | **0.44** | 44.9 |
| net4-1 | 0.78 | 3.09 | **0.53** | *2.39* |
| ckt11752_dc_1 | 0.15 | 0.63 | **0.12** | *0.57* |
| rajat23 | 0.23 | *0.47* | **0.19** | *0.47* |
| rajat22 | **0.06** | *0.16* | **0.05** | *0.15* |
| gupta2 | 5.74 | *5.98* | **0.32** | 6.37 |
| gupta1 | 2.84 | *2.06* | **0.11** | *2.07* |
| a0nsdsil | **0.09** | *0.35* | **0.08** | 0.39 |
| blockqp1 | **0.04** | *0.38* | **0.04** | *0.38* |
| trans5 | 0.22 | 0.70 | **0.16** | *0.68* |
| ins2 | 0.53 | *1.59* | **0.34** | *1.59* |
| rajat29 | 3.32 | 11.7 | **2.25** | *9.93* |

Table 4: Comparison of the times and the estimated number of reals in $L$ ($\times 10^6$) generated by orderings from QAMD and AMDD algorithms.

matrix with mean external degree

$$\frac{\mu_1 n_1 - 2d_i}{n_1 - 1}.$$

Our proposed new variant, which we call the AMDD method, is outlined in Algorithm 3. The AMDD method is simple to implement and uses the same amount of memory as the classical AMD algorithm.

In Figure 1, we observe that the time for AMDD to calculate a pivot order for problem (1) increases with $m$ at roughly the same rate as for the QAMD method. The key point is that, as $m$ increases, there is only a small drop in the number of rows detected as dense by AMDD (Figure 2).

In Table 4, we compare the time to compute the elimination ordering and the estimated number of reals in $L$ generated by orderings from the QAMD and AMDD algorithms. We observe that, in general, AMDD is significantly faster than QAMD and the quality of the ordering is often better. Note that, if we apply AMDD to the random banded matrix that we reported on in Section 3.3, then only 39 rows/columns are classed as dense and the results are comparable to those of the classical AMD and QAMD methods.

# 5 Conclusions

We have compared a number of variants of AMD that aim to efficiently compute elimination orderings for matrices containing some dense rows and shown that both CS_AMD and QAMD can be slow compared to AMDpre. Although AMDpre performs well on the majority of our test problems, we have shown that it is possible to construct problems where it fails to correctly detect the dense rows. This led us to propose a new variant, called AMDD, that combines the robustness of QAMD in detecting dense rows with the speed of AMDpre. Our implementation of AMDD requires no extra storage over that required in the implementation of the classical AMD algorithm used within this paper and is much more straightforward than that of QAMD because it does not need to restart or sub-partition the dense rows. The current variant of AMD implemented within the `HSL` package `MC47` will be replaced by the AMDD variant.

# Acknowledgments

We are grateful to Patrick Amestoy for helpful discussions on QAMD and to John Reid for several discussions on implementing minimum degree algorithms and for his careful reading of this paper.

# References

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

[2] P. R. Amestoy, H. S. Dollar, J. K. Reid, and J. A. Scott, *An approximate minimum degree algorithm for matrices with dense rows*, Tech. Rep. RAL-TR-2007-020, Rutherford Appleton Laboratory, 2007.

[3] J. L. Carmen, *An AMD preprocessor for matrices with some dense rows and columns.* http://www.netlib.org/linalg/amd/amdpre.ps.

[4] T. Davis, *The University of Florida Sparse Matrix Collection*, 2007. http://www.cise.ufl.edu/davis/techreports/matrices.pdf.

[5] T. A. Davis, *Direct methods for sparse linear systems*, vol. 2 of Fundamentals of Algorithms, SIAM, Philadelphia, PA, 2006.

[6] I. Duff, *MA57– a new code for the solution of sparse symmetric definite and indefinite systems*, ACM Transactions on Mathematical Software, 30 (2004), pp. 118–154.

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

[8] I. Duff and J. Scott, *Towards an automatic ordering for a symmetric sparse direct solver*, Tech. Rep. RAL-TR-2006-001, Rutherford Appleton Laboratory, 2006.

[9] A. George, *Nested dissection of a regular finite-element mesh*, SIAM J. Num. Anal., 10 (1973), pp. 345–363.

[10] HSL, *A collection of Fortran codes for large-scale scientific computation.* See http://hsl.rl.ac.uk/, 2007.

[11] MUMPS, *Multifrontal Massively Parallel Solver.* See http://mumps.enseiht.fr or http://graal.ens-lyon.fr/MUMPS/.

[12] D. J. Rose, *Symmetric elimination on sparse positive definite systems and the potential flow network problem*, PhD thesis, Harvard University, 1970.

[13] ——, *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, 1973, pp. 183–217.

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

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