NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2010; **17**:43–55
Published online 9 April 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla.647

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

H. S. Dollar*, † and J. A. Scott

Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, U.K.

SUMMARY

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 potential limitations. This leads us to propose a new variant that offers both speed and robustness. Copyright © 2009 John Wiley & Sons, Ltd.

Received 11 March 2008; Revised 2 March 2009; Accepted 3 March 2009

KEY WORDS: approximate minimum degree ordering algorithm; sparse symmetric matrices; dense rows; graph algorithms; ordering algorithms

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 the 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 $PAP^{\mathrm{T}}=LL^{\mathrm{T}}$, where P is a permutation matrix and L is lower triangular, can be computed. More generally, numerical

*Correspondence to: H. S. Dollar, Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, U.K.
†E-mail: sue.dollar@stfc.ac.uk

Contract/grant sponsor: Engineering and Physical Sciences Research Council; contract/grant number: EP/E053351/1

pivoting must be incorporated during the factorization phase to maintain numerical stability and, in this case, the pivot sequence computed by the preordering step 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 [1] and so heuristics are used to find a good ordering. Two main classes of methods are widely used: those based on nested dissection [2] and those based on the minimum degree algorithm [3]. 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) [4]. 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 (AMD) algorithm and, in particular, the AMD algorithm introduced by Amestoy *et al.* [5, 6] 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 [5].

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 a 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 that its adjacency list has to be scanned and updated, requiring $\mathcal{O}(n^2)$ operations for a problem 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 [7] and Davis [8] (see also [6]): 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` [9] and was recently discussed by Amestoy *et al.* [10] uses a more sophisticated dynamic partition of the rows into dense and sparse rows to efficiently compute the pivot order. Our experiments show 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 organized as follows. We end this section by introducing our test environment and then, in Section 2, we give a brief review of the minimum degree and AMD algorithms. In Section 3, the methods of Carmen, Davis, and Amestoy *et al.* are described and numerical results are used to illustrate their weaknesses. The new variant of the AMD algorithm that we propose to overcome these problems is introduced in Section 4 and the numerical results are presented in Section 4.1. We draw our conclusions in Section 5.

1.1. Test environment

Table I lists the test problems that we use to compare the variants of the minimum degree algorithm. The problems are from the University of Florida Sparse Matrix Collection [11]. If A has an unsymmetric sparsity pattern, we work with the sparsity pattern of $\hat{A} = A + A^{\mathrm{T}}$ (in computing the pattern of the symmetrized matrix $\hat{A}$, we ignore the exact numerical calculation and include any

DOI: 10.1002/nla

Table I. The test set: the order n, the number of off-diagonal entries nz in $\hat{A} = A + A^{\mathrm{T}}$, the maximum $d_{\max}$, the mean μ, and the standard deviation σ of the number of entries in the rows of $\hat{A}$.

	Problem	n	nz	$d_{\max}$	μ	σ	σ/μ
1	crystk03	24696	1751178	81	70.9	14.2	0.20
2	nd12k	36000	14220946	519	395	83.2	0.21
3	dawson5	51537	1010777	33	19.6	5.32	0.27
4	ramage02	16830	2866352	270	170	52.9	0.31
5	heart1	3557	1384216	1119	389	124	0.32
6	cegb2919	2919	321543	267	110	41.6	0.38
7	Kuu	7102	340200	98	47.9	20.0	0.42
8	crankseg_1	52804	10614210	2703	201	88.7	0.44
9	gyro_k	17361	1021159	360	58.8	32.1	0.54
10	3dtube	45330	3213618	2364	70.9	39.7	0.56
11	invextr1_new	30412	1844242	257	60.6	40.4	0.67
12	pattern1	19242	9323432	6028	485	380	0.78
13	lpl1	32460	328036	253	10.1	15.5	1.53
14	gupta3	16783	9323427	14672	556	1234	2.22
15	mip1	66463	10352819	66395	156	351	2.25
16	av41092	41092	3366164	2661	81.9	193	2.36
17	net4-1	88343	2441727	4791	27.6	85.3	3.09
18	ckt11752_dc_1	49702	337616	2921	6.79	24.7	3.63
19	rajat23	110355	454986	3401	4.12	19.4	4.71
20	rajat22	39899	162996	3401	4.09	25.0	6.12
21	gupta2	62064	4248286	8413	68.5	356	5.20
22	gupta1	31802	2164210	8413	68.1	360	5.30
23	Chebyshev4	80016	355034	5003	4.44	50.0	11.3
24	a0nsdsil	60012	640033	40011	10.7	305	28.6
25	case39	40216	1042160	20024	25.9	316	12.2
26	blockqp1	60012	640033	40011	10.7	305	28.6
27	trans5	116835	744236	114190	6.37	387	60.8
28	ins2	309412	2751484	309412	8.89	590	66.4
29	rajat29	643994	5562244	454745	8.64	785	90.8

explicit zeros). For each example, we give the order n of A and the number nz of off-diagonal entries in the pattern of $\hat{A}$. Furthermore, defining d_i to be the number of off-diagonal entries in the ith row of $\hat{A}$, Table I includes the maximum $d_{\max}$, the mean μ, and the standard deviation σ of the d_i. As in [10], the problems are presented in increasing order of σ/μ.

Our numerical experiments are performed on a 3.6 GHz Intel Xeon dual processor Dell Precision 670 with 4 Gbytes of RAM, running Red Hat Enterprise Linux Server release 5.1 (kernel 2.6.18-53.1.13.el5). All codes used in this paper are written in Fortran and the g95 compiler with the -O4 option is used.

The statistics we use when comparing AMD variants are the CPU time (in seconds) required to compute the pivot sequence and the forecast number $nz(L)$ of reals in the matrix factor. The latter provides a measure of the quality of the ordering and is generated by passing the pivot order to the analyse phase of the HSL [12] sparse direct solver MA57 [13] (Version 3.2.0). The input order in all tests is as supplied but note that tie breaking within each algorithm can have some (usually minor) effect on the final ordering.

DOI: 10.1002/nla

2. THE MINIMUM DEGREE AND AMD ALGORITHMS

We start by briefly recalling the minimum degree and AMD algorithms for computing a pivot ordering for a sparse symmetric matrix and introduce the notation and terminology we will use throughout this paper.

The minimum degree and AMD algorithms may be presented using the graph model of Rose [14, 15]. The nonzero pattern of a sparse symmetric matrix $A=\{a_{ij}\}$ of order n can be represented by an undirected graph $G^0=(V^0, E^0)$ 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 in V^0 are *adjacent* to each other (neighbours) in graph G^0 if the edge (i, j) is present in E^0.

The elimination graph $G^k=(V^k, E^k)$ 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 *degree* of node i is defined to be the number of nodes adjacent to node i in G^k. We denote the degree by d_i^k. At each stage, the minimum degree algorithm chooses the kth pivot to be a node p of minimum degree in G^{k-1}. Algorithm 1 gives an outline of the minimum degree algorithm. We use the term *variable* for a node that has not been removed from the elimination graph.

Algorithm 1 The minimum degree algorithm

Let $G^0=(V^0, E^0)$ be the undirected graph associated with an $n\times n$ symmetric matrix.
Compute the degree d_i^0 of each $i\in V^0$.
Initialize $k=1$.
while $k\leqslant n$ **do**
  Select variable $p\in V^{k-1}$ to minimize d_p^{k-1}.
  Construct $G^k=(V^k, E^k)$ (see below).
  Update d_i^k for each variable i that is adjacent to p in G^{k-1}.
  Set $k\leftarrow k+1$.
end while

The graph G^k depends on G^{k-1} and the choice of the kth pivot. G^k is constructed by selecting the kth 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. However, the calculation of the degree of a variable may then be expensive. To improve the efficiency, Amestoy *et al.* [5, 6] proposed the AMD algorithm, which calculates an upper bound, $\bar{d}_i$, on the degree of a variable and uses this when choosing the next pivot. Numerical results have shown that the AMD algorithm produces orderings that are comparable in quality to the best classical minimum degree algorithms while being significantly faster [5].

When a pivot p is eliminated, the AMD algorithm of Amestoy *et al.* (which we will refer to throughout the remainder of this paper as the *classical AMD algorithm*) updates the upper bounds $\bar{d}_i$ of all the nodes adjacent to p. If the row corresponding to node i is (almost) dense, it is (almost) certainly adjacent to p and updating $\bar{d}_i$ will involve a large number of comparisons. Thus (almost) every step is expensive, making the classical AMD algorithm very inefficient when the matrix A has some dense (or almost dense) rows and columns. This was demonstrated for a range of practical problems in [10] and is also illustrated in Table II for a subset of our test

DOI: 10.1002/nla

Table II. Comparison of the times (in seconds) and the predicted number of entries in L ($\times 10^6$) generated by orderings from the minimum degree algorithm and the classical AMD algorithm.

	Minimum degree		Classical AMD	
Problem	Time	$nz(L)$	Time	$nz(L)$
lpl1	1.21	0.97	0.27	0.97
mip1	41.2	38.9	11.4	39.0
gupta2	1931	5.86	92.3	5.89
gupta1	391	2.02	32.5	2.06
blockqp1	47.1	0.38	6.71	0.38
trans5	362	0.68	212	0.68
ins2	2436	1.53	791	1.53

problems. We use a Fortran 95 version of the minimum degree algorithm, which was originally developed for the earlier solver MA27 [16]. For the classical AMD algorithm, we use the HSL code MC47 with its control parameter ICNTL(4) set to -1. These results show that, as expected, the AMD algorithm is significantly more efficient than the minimum degree algorithm, while the quality of the orderings is comparable. For the gupta problems, the times differ by at least an order of magnitude. However, for a number of our test problems with dense rows (including gupta2, trans5, and ins2), the AMD times are still high, that is, they are significantly greater when compared with problems of similar size and density but with no dense rows (see [10]). The algorithms described in Section 3 aim to reduce these times while maintaining ordering quality.

3. AMD VARIANTS FOR 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 [7], Davis [8], and Amestoy *et al.* [10]. 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) implements the CS_AMD algorithm.

3.1. The AMDpre and CS_AMD algorithms

The AMDpre [7] and CS_AMD [8] algorithms (see also the C version of AMD in [6]) 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

$$A = \left[\begin{array}{c|c} A_1 & A_r^T \\ \hline A_r & A_d \end{array}\right] \tag{1}$$

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

DOI: 10.1002/nla

The CS_AMD algorithm classifies a row in A as dense if its degree is greater than $\max(16, \alpha\sqrt{n})$, 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 [12] sparse solver MA27 [16].

The AMDpre algorithm uses a procedure that is equivalent to that given in Algorithm 2. The threshold γ 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 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.

Algorithm 2 The AMDpre method for choosing A_1

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

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 [9]. 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 [10], Amestoy *et al.* define a row to be full if all of its entries are (symbolically) nonzero; a row is quasi-dense if it has a high proportion of nonzero entries (so that its degree is large); a row is sparse if it is neither full nor quasi-dense. Amestoy *et al.* begin by partitioning the matrix A into the form

$$A = \left[\begin{array}{c|c|c} A_{\mathrm{s}} & A_{\mathrm{q1}}^{\mathrm{T}} & A_{\mathrm{f1}}^{\mathrm{T}} \\ \hline A_{\mathrm{q1}} & A_{\mathrm{q}} & A_{\mathrm{f2}}^{\mathrm{T}} \\ \hline A_{\mathrm{f1}} & A_{\mathrm{f2}} & A_{\mathrm{f}} \end{array}\right]$$

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 starts by applying the AMD algorithm to the submatrix A_{s} but if as the eliminations are performed, a row that was initially sparse becomes quasi-dense or full, its variable is removed from A_{s} and placed into A_{q} or A_{f}, respectively. When all the variables in A_{s} have been either eliminated or reclassified, the exact degrees of the quasi-dense rows are calculated and these rows are then reclassified as either sparse or full. If k pivots have already been chosen, the reduced matrix $A^{(k)}$ of order $n-k$ is of the form

$$A^{(k)} = \left[\begin{array}{c|c} A_{\mathrm{s}}^{k} & (A_{\mathrm{f1}}^{k})^{\mathrm{T}} \\ \hline A_{\mathrm{f1}}^{k} & A_{\mathrm{f}}^{k} \end{array}\right]$$

DOI: 10.1002/nla

where the rows in A_s^k are sparse and those in A_f^k are full. The QAMD algorithm restarts by applying the AMD algorithm to A_s^k and, again, reclassifying rows as quasi-dense or full when appropriate. 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 [10], a variable i is reclassified as quasi-dense if it is not known to be full and $\bar{d}_i^s + n_q + n_f \geqslant \tau$, where n_q and n_f are the numbers of quasi-dense and dense rows, respectively, and $\bar{d}_i^s$ 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 τ is given in [10].

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 during each restart.

3.3. A comparison of AMD variants

In this section, we present numerical results to demonstrate that, although the variants of AMD introduced so far outperform the classical AMD algorithm, problems remain on which they perform poorly. For our experiments with the classical AMD and QAMD algorithms, we use the HSL code `MC47` with its control parameter `ICNTL(4)` set to -1 and 1, respectively. We use the Fortran implementation of AMDpre provided at http://www.netlib.org/linalg/amd/, but, for consistency, have replaced the call to `amdbar` with a call to `MC47` with its control parameter `ICNTL(4)` set to -1. We use our own Fortran implementation of the CS_AMD algorithm. We remark that we have also run the Matlab implementation of CS_AMD; in our tests our implementation gave orderings of comparable quality.

In Table III, we report the results for our test problems. For most of our test problems, significant time gains are achieved by using an AMD variant that allows for some of the rows being dense. `rajat29` is a notable example: the minimum degree algorithm fails to compute a pivot order within 10 000 seconds, classical AMD requires 4115 seconds, whereas the dense row variants take no more than 3.5 seconds. Other examples for which the dense row variants achieve significant time savings include the `gupta` problems, `trans5`, and `ins2`.

For some problems, including the `gupta` problems, AMDpre is much faster than QAMD, without compromising on quality. The gain in speed is because AMDpre only needs to store the structure of the submatrix A_1 (see (1)) but QAMD requires the whole structure to be stored and occasionally searched to allow the method to restart correctly. AMDpre is also generally faster than CS_AMD. This is because CS_AMD can fail to detect and remove some of the rows that should be categorized as dense and these remaining dense rows lead to slow run times. In particular, Algorithm 2 will fail to correctly detect all the dense rows in a matrix if the maximum degree is smaller than $\beta\sqrt{n}$ but still significantly larger than the average degree of the resulting matrix A_1. For example, for `gupta2`, CS_AMD flags only 258 dense rows while AMDpre finds 1193, leading to the CS_AMD time being almost 40 times slower than that of AMDpre.

Of course, we not only want an algorithm that is fast, we also require an algorithm that produces orderings that lead to sparse factors. In terms of ordering quality, CS_AMD and QAMD generally produce orderings of comparable quality to the classical AMD algorithm (although for a small number of examples, including `gupta3` and `rajat29`, the factor obtained using the QAMD is

DOI: 10.1002/nla

Table III. Comparison of the times (in seconds) and the predicted number of reals in L ($\times 10^6$) generated by orderings from the minimum degree (MinDeg), classical AMD (Classical), AMDpre, CS_AMD, QAMD and AMDD algorithms.

Problem		MinDeg	Classical	AMDpre	CS_AMD	QAMD	AMDD
crystk03	Time	0.13	0.09	0.09	0.10	0.09	0.11
	$nz(L)$	13.6	11.9	11.9	11.9	11.9	11.9
nd12k	Time	5.92	1.52	0.71	1.59	1.47	1.70
	$nz(L)$	178	156	308	156	156	156
dawson5	Time	0.26	0.20	0.20	0.21	0.21	0.25
	$nz(L)$	5.44	4.62	4.62	4.62	4.62	4.62
ramage02	Time	0.15	0.11	0.10	0.12	0.11	0.13
	$nz(L)$	21.0	19.3	29.8	19.3	19.3	19.3
heart1	Time	0.05	0.04	0.04	0.04	0.04	0.05
	$nz(L)$	1.59	1.61	3.36	1.51	1.61	1.51
cegb2919	Time	0.01	0.01	0.01	0.01	0.01	0.01
	$nz(L)$	0.35	0.34	1.03	0.34	0.34	0.34
Kuu	Time	0.02	0.01	0.01	0.01	0.01	0.02
	$nz(L)$	0.39	0.38	0.54	0.38	0.38	0.38
crankseg_1	Time	0.55	0.43	0.42	0.47	0.42	0.50
	$nz(L)$	49.8	38.9	75.4	38.3	38.9	45.6
gyro_k	Time	0.07	0.05	0.06	0.06	0.05	0.07
	$nz(L)$	1.27	1.23	1.59	1.23	1.23	1.25
3dtube	Time	0.25	0.18	0.19	0.19	0.18	0.22
	$nz(L)$	32.4	27.3	28.2	28.2	27.3	28.2
invextr1_new	Time	0.41	0.21	0.22	0.22	0.21	0.26
	$nz(L)$	17.3	14.8	16.8	14.8	14.8	14.8
pattern1	Time	4.21	0.86	0.30	0.64	0.86	0.51
	$nz(L)$	48.4	48.4	61.3	48.4	48.4	54.4
lpl1	Time	1.21	0.27	0.27	0.28	0.19	0.35
	$nz(L)$	0.97	0.97	0.99	0.97	1.20	0.99
gupta3	Time	95.9	9.31	0.23	0.63	3.68	0.29
	$nz(L)$	5.72	5.72	5.60	5.35	6.52	5.53
mip1	Time	41.2	11.4	0.48	1.08	0.85	0.68
	$nz(L)$	38.9	39.0	44.8	39.3	39.7	44.1
av41092	Time	361	5.65	2.56	5.37	6.48	3.24
	$nz(L)$	174	174	167	177	181	170
net4-1	Time	6.09	2.15	0.53	0.59	0.80	0.64
	$nz(L)$	2.46	2.38	2.39	2.37	3.09	2.39
ckt11752_dc_1	Time	0.47	0.20	0.12	0.18	0.16	0.15
	$nz(L)$	0.59	0.56	0.57	0.56	0.63	0.57
rajat23	Time	0.62	0.33	0.19	0.28	0.24	0.21
	$nz(L)$	0.45	0.46	0.47	0.46	0.47	0.47
rajat22	Time	0.27	0.13	0.05	0.07	0.08	0.07
	$nz(L)$	0.15	0.15	0.15	0.15	0.16	0.15
gupta2	Time	1931	92.3	0.41	13.0	5.43	0.45
	$nz(L)$	5.86	5.89	6.30	5.77	5.98	6.41
gupta1	Time	391	32.5	0.11	3.05	2.66	0.15
	$nz(L)$	2.02	2.06	2.08	2.00	2.06	2.07
Chebyshev4	Time	17.8	3.50	0.36	1.03	0.68	0.44
	$nz(L)$	21.9	21.3	25.2	21.3	26.7	22.0
a0nsdsil	Time	13.1	2.93	0.07	0.07	0.10	0.10
	$nz(L)$	0.34	0.34	0.39	0.39	0.35	0.39

DOI: 10.1002/nla

Table III. *Continued*.

Problem		MinDeg	Classical	AMDpre	CS_AMD	QAMD	AMDD
blockqp1	Time	47.1	6.71	0.04	0.04	0.05	0.05
	$nz(L)$	0.38	0.38	0.38	0.38	0.38	0.38
trans5	Time	362	212	0.22	0.21	0.28	0.24
	$nz(L)$	0.68	0.68	0.69	0.69	0.70	0.68
ins2	Time	2436	791	0.35	0.33	0.53	0.44
	$nz(L)$	1.53	1.53	1.59	1.59	1.59	1.59
rajat29	Time	$>10^4$	4115	2.30	2.61	3.25	2.58
	$nz(L)$	—	9.77	9.97	9.87	11.7	9.93

about 20% denser than that obtained using the classical AMD). However, a serious drawback with AMDpre is that it can remove a large number of rows that the other variants classify as sparse and this results in the quality of the ordering being lost. This is illustrated by the example nd12k for which CS_AMD and AMDpre remove 0 and 17 155 rows, respectively. In this case, $nz(L)$ for AMDpre is twice that of the other AMD variants.

The results of our numerical experiments suggest 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

In this section, we introduce a new AMD variant that is designed to overcome the weaknesses of the existing variants that were highlighted in the previous section. Let us assume that k rows/columns have been classified as dense and that A_k is the matrix that remains after these k rows and columns have been removed from A. Let μ_k be the mean of the degrees of the variables in A_k and i be a dense row in A_k. If row and column i are removed from A_k, we expect the mean value of the degrees of the remaining 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 μ_k with the value that μ_{k+1} would take if row/column i are removed. We declare the row/column as dense if the difference is greater than a threshold t_{n-k}. Numerical experimentation on a wide range of problems has shown that defining t_{n-k} as

$$t_{n-k}=\frac{\delta\ln(n-k)}{n-k}+\frac{\mu_k}{n-k-1} \tag{2}$$

with $\delta=40$ 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 δ: values between 30 and 50 produced, in general, similar results.

Note that removing row and column i from A_k results in a matrix A_{k+1} with mean degree

$$\mu_{k+1}=\frac{(n-k)\mu_k-2d_{ki}}{n-k-1}$$

DOI: 10.1002/nla

Algorithm 3 The AMDD method

Set $A_0 = A$, $k = 0$, and q to be an empty list
For each row i in A_0, set d_{0i} to be the degree of row i
Calculate the mean μ_0 of the degrees of the rows in A_0
Calculate t_n, where t_n is defined by (2)
Select row i in A_0 of largest degree
while $\mu_k - ((n-k)\mu_k - 2d_{ki})/(n-k-1) \geqslant t_{n-k}$ **do**
  Add i to the beginning of the list q
  Set $A_{k+1} \leftarrow A_k$ with row and column i removed
  $\mu_{k+1} = ((n-k)\mu_k - 2d_{ki})/(n-k-1)$
  Set $k \leftarrow k+1$
  For each row i in A_k, calculate d_{ki}
  Calculate t_{n-k}, where t_{n-k} is defined by (2)
  Select row i in A_k of largest degree
end while
Apply classical AMD to A_k to form a pivot order
Append the list q to the end of this pivot order to form a pivot order for A

where d_{ki} is the number of off-diagonal entries in the ith row of A_k. Therefore, our criteria for declaring a row as dense are

$$\mu_k - \frac{(n-k)\mu_k - 2d_{ki}}{n-k-1} \geqslant \frac{\delta \ln(n-k)}{n-k} + \frac{\mu_k}{n-k-1}$$

Rearranging, we obtain

$$d_{ki} - \mu_k \geqslant \frac{\delta}{2}\frac{n-k-1}{n-k}\ln(n-k) \approx \frac{\delta}{2}\ln(n-k)$$

Comparing our definition of a dense row with that of AMDpre, we observe that our new definition additionally takes the mean degree into account, it contains a function that rises less steeply with the order of the matrix, and it also makes allowances for the reduction in the order as rows are removed. We believe that all of these properties are sensible. Our proposed new variant, which we call the AMDD method, is outlined in Algorithm 3. Note that, given an efficient implementation of the classical AMD, the AMDD variant is straightforward to implement and requires no additional memory.

4.1. Numerical results for the AMDD algorithm

In this section, we compare our proposed AMDD variant with the AMDpre, CS_AMD, and QAMD variants. Table III contains the complete set of numerical results. In Figure 1, we compare the forecast number of reals in the matrix factor relative to that of the AMDD variant. The problems are numbered as in Table I. For seven out of the first nine problems, AMDpre removed a large number of rows that the other variants regarded as sparse and, as a result, the quality of the ordering was significantly worse than that for all the other variants. By comparison, AMDD consistently produces high-quality orderings. Problem 8 (`crankseg_1`) is a notable exception where both CS_AMD and QAMD result in the forecast number of reals in the matrix factors being 38.3×10^6 and 38.9×10^6, respectively, but AMDD has a corresponding value of 45.6×10^6. However, it is

DOI: 10.1002/nla

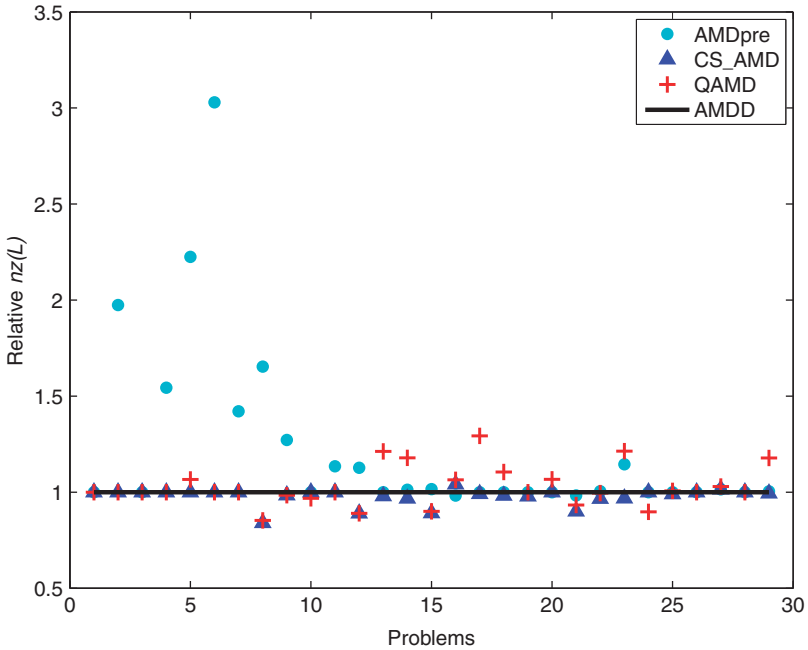

Figure 1. The forecast number of reals $nz(L)$ in the matrix factor relative to that of AMDD. The problems are numbered as in Table I.

worth noting that AMDD produced a higher-quality ordering than the minimum degree algorithm ($nz(L) = 49.8 \times 10^6$).

As noted in Section 3.3, we also require the AMDD variant to be efficient. In Figure 2, we compare the time required for CS_AMD and QAMD to form an ordering relative to the time taken by AMDD. If the forecast number of nonzeros in the matrix factor exceeds 1.25 times that of the AMDD, then we omit the (relative) time from the figure. We also omit the relative times that are greater than 2.4 and those for which the time differs from that of the AMDD by at most 0.03 seconds. We observe that the fastest time for problem 13 was achieved by QAMD, but it gave a poorer-quality ordering than AMDD. We see that AMDD is significantly faster than CS_AMD and QAMD for a number of our test problems.

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 with AMDpre. Although AMDpre performs well on many problems, we have shown that it can fail 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

DOI: 10.1002/nla

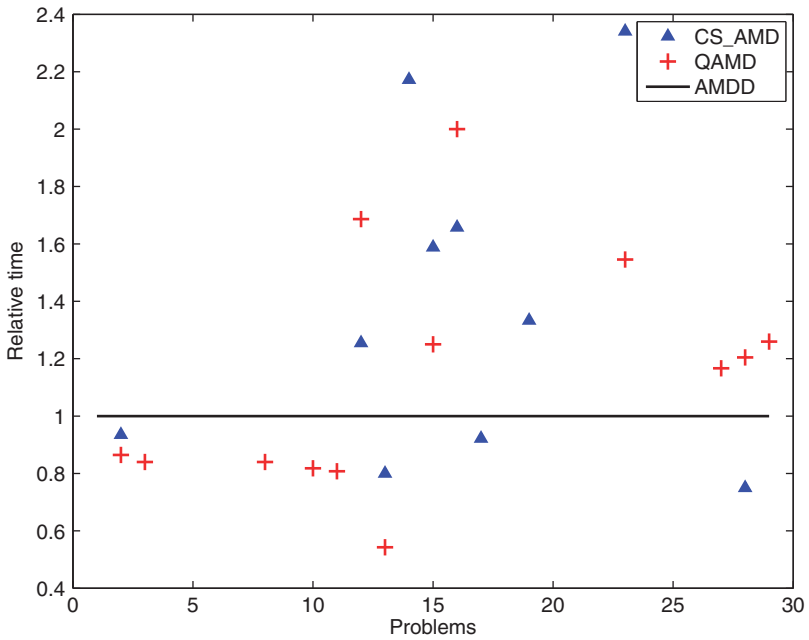

Figure 2. The time required to compute the ordering relative to that of AMDD. Times are omitted from the figure if the relative value of $nz(L)$ (with respect to AMDD) exceeds 1.25, if the relative time is greater than 2.4, or if the time differs from that of AMDD by at most 0.03 seconds. The problems are numbered as in Table I.

straightforward to implement than QAMD because it does not need to restart or sub-partition the dense rows.

An efficient Fortran 95 implementation of the AMDD is available as one of the options offered by the new ordering package `HSL_MC68`. We remark that this package also offers options for a variety of different orderings, allowing the user a straightforward way of generating and then comparing the different orderings for his or her applications. `HSL_MC68` is available as part of the 2007 release of the mathematical software library HSL. All use of HSL requires a licence. Individual HSL packages (together with their dependencies and accompanying documentation) are available without charge to individual academic users for their personal (non-commercial) research and for teaching; licences for other uses involve a fee. Details of all HSL packages and how to obtain a licence plus conditions of use are available at www.cse.scitech.ac.uk/nag/hsl/.

ACKNOWLEDGEMENTS

We are grateful to Patrick Amestoy for the helpful discussions on QAMD and to John Reid for several discussions on implementing the minimum degree algorithms and for his careful reading of this paper. We would also like to thank the referees for their constructive comments.

DOI: 10.1002/nla

REFERENCES

1. Yannakakis M. Computing the minimum fill-in is NP-complete. *SIAM Journal on Discrete Mathematics* 1981; **2**:77–79.
2. George A. Nested dissection of a regular finite-element mesh. *SIAM Journal on Numerical Analysis* 1973; **10**:345–363.
3. Tinney W, Walker J. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE* 1967; **55**:1801–1809.
4. Duff IS, Scott JA. Towards an automatic ordering for a symmetric sparse direct solver. *Technical Report RAL-TR-2006-001*, Rutherford Appleton Laboratory, 2006.
5. Amestoy PR, Davis TA, Duff IS. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications* 1996; **17**(4):886–905.
6. Amestoy PR, Davis TA, Duff IS. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software* 2004; **30**(3):381–388.
7. Carmen JL. An AMD preprocessor for matrices with some dense rows and columns. See http://www.netlib.org/linalg/amd/amdpre.ps.
8. Davis TA. *Direct Methods for Sparse Linear Systems*, *Fundamentals of Algorithms*, vol. 2. SIAM: Philadelphia, PA, 2006.
9. MUMPS. Multifrontal massively parallel solver. See http://mumps.enseiht.fr or http://graal.ens-lyon.fr/MUMPS/.
10. Amestoy PR, Dollar HS, Reid JK, Scott JA. An approximate minimum degree algorithm for matrices with dense rows. *Technical Report RAL-TR-2007-020*, Rutherford Appleton Laboratory, 2007.
11. Davis T. The University of Florida Sparse Matrix Collection, 2007. See http://www.cise.ufl.edu/davis/techreports/matrices.pdf.
12. HSL. A collection of Fortran codes for large-scale scientific computation, 2007. See http://hsl.rl.ac.uk/.
13. Duff IS. MA57—a new code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software* 2004; **30**:118–154.
14. Rose DJ. Symmetric elimination on sparse positive definite systems and the potential flow network problem. *Ph.D. Thesis*, Harvard University, 1970.
15. Rose DJ. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, Read RC (ed.). Academic Press: New York, 1973; 183–217.
16. Duff IS, Reid JK. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software* 1983; **9**(3):302–325.

DOI: 10.1002/nla