Algorithm 8xx: AMD, an approximate minimum degree ordering algorithm

Patrick R. Amestoy[*] Timothy A. Davis[†] Iain S. Duff[‡]

May 6, 2003

Abstract

AMD is a set of routines for permuting sparse matrices prior to numerical factorization, using the approximate minimum degree ordering algorithm. There are versions written in both C and Fortran 77. A MATLAB interface is included.

Categories and Subject Descriptors: G.1.3 [Numerical Analysis]: Numerical Linear Algebra – *linear systems (direct methods), sparse and very large systems* G.4 [Mathematics of Computing]: Mathematical Software – *algorithm analysis, efficiency*

General terms: Algorithms, Experimentation, Performance.

Keywords: sparse matrices, linear equations, ordering methods, minimum degree

1 Overview

AMD is a set of routines for preordering a sparse matrix prior to numerical factorization. It uses an approximate minimum degree ordering algorithm to find a permutation matrix $\mathbf{P}$ so that the Cholesky factorization $\mathbf{PAP}^{\mathsf{T}} = \mathbf{LL}^{\mathsf{T}}$ has fewer (often much fewer) nonzero entries than the Cholesky factorization of $\mathbf{A}$.

In the C version of AMD, the elimination phase is followed by an elimination tree post-ordering. This has no effect on the number of nonzeros in $\mathbf{L}$, but reorganizes the ordering so that the subsequent numerical factorization is more efficient. It also includes a pre-processing phase in which it handles "dense" rows and columns with many nonzeros. If it is given an

[*]ENSEEIHT-IRIT, 2 rue Camichel 31017 Toulouse, France. email: amestoy@enseeiht.fr. http://www.enseeiht.fr/~amestoy.

[†]Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. http://www.cise.ufl.edu/~davis. This work was supported by the National Science Foundation, under grants ASC-9111263, DMS-9223088, and DMS-0203270. Portions of the work were done while on sabbatical at Stanford University and Lawrence Berkeley National Laboratory (with funding from Stanford University and the SciDAC program).

[‡]Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX, England. email: i.s.duff@rl.ac.uk. http://www.numerical.rl.ac.uk/people/isd/isd.html. This work was supported by the the EPSRC under grant GR/R46441.

unsymmetric matrix, it operates on the symmetric nonzero pattern formed from the Boolean summation of $\mathbf{A}$ with $\mathbf{A}^\mathsf{T}$.

The two Fortran versions of AMD are identical to two versions of the AMD algorithm discussed in an earlier paper [1] (approximate minimum external degree, both with and without aggressive absorption). Details of the method used in AMD can be found in that paper. For a discussion of the long history of the minimum degree algorithm, see [2].

2 Availability

In addition to appearing as a Collected Algorithm of the ACM, AMD Version 1.0 is available at http://www.cise.ufl.edu/research/sparse. The Fortran version is available as the routine `MC47` in HSL (formerly the Harwell Subroutine Library) [3].

3 Using AMD in MATLAB

The simplest way to use AMD is within MATLAB. Once the AMD mexFunction is compiled and installed, the MATLAB statement `p = amd (A)` computes a permutation vector `p` so that the Cholesky factorization `chol(A(p,p))` is typically sparser than `chol(A)`.

An optional input argument can be used to modify the control parameters for AMD (aggressive absorption and dense row/column handling). An optional output argument provides statistics on the ordering, including an analysis of the fill-in and floating-point operation count of a subsequent factorization. AMD will print these statistics if you turn on the MATLAB sparse matrix monitor flag with `spparms ('spumoni',1)`.

4 Using AMD in a C program

The C-callable AMD library consists of four user-callable routines and one include file. There are two versions of each of the routines, with `int` and `long` integers.

- `amd_order` (`long` version: `amd_l_order`)

  Computes the approximate minimum degree ordering of an n-by-n matrix $\mathbf{A}$. Returns a permutation vector `P` of size `n`, where `P [k] = i` if row and column `i` are the `k`th row and column in the permuted matrix. This routine allocates its own memory, and requires $O(|\mathbf{A}|)$ space, where $|\mathbf{A}|$ is the number of nonzero entries in the matrix. It computes statistics about the matrix $\mathbf{A}$, such as the symmetry of its nonzero pattern, the number of nonzeros in $\mathbf{L}$, and the floating-point operations required for Cholesky and LU factorizations. The user's input matrix is not modified.

- `amd_defaults` (`long` version: `amd_l_defaults`)

  Sets the default control parameters in the `Control` array. These can then be modified as desired before passing the array to the other AMD routines.

- `amd_control` (`long` version: `amd_l_control`)

  Prints the control parameters.

- `amd_info` (`long` version: `amd_l_info`)

  Prints the statistics computed by AMD.

The nonzero pattern of the matrix **A** is represented in compressed column form, which is identical to the sparse matrix representation used by MATLAB. It consists of two arrays, where the matrix is `n`-by-`n`, with `nz` entries. For the `int` version of AMD, the two arrays are defined as:

```
    int Ap [n+1] ;
    int Ai [nz] ;
```

The row indices of entries in column `j` are stored in `Ai[Ap[j]` ... `Ap[j+1]-1]`. No duplicate row indices may be present, and the row indices in any given column must be sorted in ascending order. The first entry `Ap[0]` must be zero. The total number of entries in the matrix is thus `nz = Ap[n]`. The matrix must be square, but it does not need to be symmetric. The diagonal entries may be present, but are ignored. AMD checks the input matrix and returns an error code if it is invalid. The follow program illustrates the basic usage of AMD.

```
#include <stdio.h>
#include "amd.h"

int n = 5 ;
int Ap [ ] = { 0,   2,       6,       10,  12, 14} ;
int Ai [ ] = { 0,1, 0,1,2,4, 1,2,3,4, 2,3, 1,4   } ;
int P [5] ;

int main (void)
{
    int k ;
    (void) amd_order (n, Ap, Ai, P, (double *) NULL, (double *) NULL) ;
    for (k = 0 ; k < n ; k++) printf ("P [%d] = %d\n", k, P [k]) ;
    return (0) ;
}
```

The `Ap` and `Ai` arrays represent the binary matrix

$$\mathbf{A} = \left[\begin{array}{ccccc} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array} \right].$$

AMD constructs the pattern of $\mathbf{A} + \mathbf{A}^\mathsf{T}$, and returns a permutation vector of $(0, 3, 1, 4, 2)$. More example programs are included with the AMD package.

5 Using AMD in a Fortran program

Two Fortran versions of AMD are provided. The `AMD` routine computes the approximate minimum degree ordering, using aggressive absorption. The `AMDBAR` routine is identical, except that it does not perform aggressive absorption. The `AMD` routine is essentially identical to the HSL routine `MC47B/BD`.

The Fortran versions differ from the C routines. The AMD algorithms were originally coded in Fortran and so are identical to the routines used in the experimental results in [1]. The internal routines require a symmetric nonzero pattern, with no diagonal entries present although the `MC47A/AD` wrapper in HSL allows duplicates, ignores out-of-range entries, and only uses entries from the upper triangular part of the matrix. Although we have an experimental Fortran code for treating "dense" rows, the Fortran codes in this release do not treat "dense" rows and columns of **A** differently, and thus their run time can be high if there are a few dense rows and columns in the matrix. They do not perform a post-ordering of the elimination tree, compute statistics on the ordering, or check the validity of their input arguments. These functions are provided by the HSL routines `MA57L/LD` and `MC47A/AD`, which are not part of this release. However, details on an assembly tree that respects the AMD ordering are returned from the calls to the Fortran codes. Only one `integer` version of each routine is provided. Both Fortran versions overwrite the user's input matrix, in contrast to the C version. The two Fortran versions have the same calling sequence, and only differ in the name of the routine.

```
        INTEGER N, IWLEN, PFREE, NCMPA, IW (IWLEN), PE (N), DEGREE (N), NV (N),
     $          NEXT (N), LAST (N), HEAD (N), ELEN (N), W (N), LEN (N)

        CALL AMD (N, PE, IW, LEN, IWLEN, PFREE, NV, NEXT,
     $          LAST, HEAD, ELEN, DEGREE, NCMPA, W)
```

The input matrix is provided to `AMD` and `AMDBAR` in three arrays, `PE`, of size `N`, `LEN`, of size `N`, and `IW`, of size `IWLEN`. The size of `IW` must be at least `NZ+N`. The recommended size is `1.2*NZ + N`. On input, the indices of nonzero entries in row `I` are stored in `IW`. `PE (I)` is the index in `IW` of the start of row `I`. `LEN (I)` is the number of entries in row `I`. The matrix is 1-based, with row and column indices in the range 1 to `N`. Row `I` is contained in `IW (PE (I) ... PE (I) + LEN (I) - 1)`. The diagonal entries must not be present. The indices within each row must not contain any duplicates, but they need not be sorted. The rows themselves need not be in any particular order, and there may be empty space between the rows. If `LEN (I)` is zero, then there are no off-diagonal entries in row `I`, and `PE (I)` is ignored. The integer `PFREE` defines what part of `IW` contains the user's input matrix, which is held in `IW (1 ... PFREE-1)`. The contents of `IW` and `LEN` are undefined on output, and `PE` is modified to contain information about the ordering.

As the algorithm proceeds, it modifies the `IW` array, placing the pattern of the partially eliminated matrix in `IW (PFREE ... IWLEN)`. If this space is exhausted, the space is compressed. The number of compressions performed on the `IW` array is returned in the scalar `NCMPA`. The value of `PFREE` on output is the length of `IW` required for no compressions to be needed.

The output permutation is returned in the array `LAST`, of size `N`. If `I = LAST (K)`, then `I` is the `K`th row in the permuted matrix. The inverse permutation is returned in the array `ELEN`, where `K = ELEN (I)` if `I` is the `K`th row in the permuted matrix.

On output, the `PE` and `NV` arrays hold the assembly tree, a supernodal elimination tree that represents the relationship between columns of the Cholesky factor **L**. If `NV (I)` > 0, then `I` is a node in the assembly tree, and the parent of `I` is `-PE (I)`. If `I` is a root of the tree, then `PE (I)` is zero. The value of `NV (I)` is the number of entries in the corresponding column of **L**, including the diagonal. If `NV (I)` is zero, then `I` is a non-principal node that is not in the assembly tree. Node `-PE (I)` is the parent of node `I` in a subtree, the root of which is a node in the assembly tree. All nodes in one subtree belong to the same supernode in the assembly tree.

The other size `N` arrays (`DEGREE`, `HEAD`, `NEXT`, and `W`) are used as workspace, and are not defined on input or output.

The follow program illustrates the basic usage of the Fortran version of AMD. The `AP` and `AI` arrays represent the binary matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

in a conventional 1-based column-oriented form, except that the diagonal entries are not present. The matrix is the same as nonzero pattern of $\mathbf{A} + \mathbf{A}^\mathsf{T}$ in the C program, in Section 4. The output permutation is $(4, 1, 3, 5, 2)$.

```
      INTEGER N, NZ, J, K, P, IWLEN, PFREE, NCMPA
      PARAMETER (N = 5, NZ = 10, IWLEN = 17)
      INTEGER AP (N+1), AI (NZ), LAST (N), PE (N), LEN (N), ELEN (N),
     $      IW (IWLEN), DEGREE (N), NV (N), NEXT (N), HEAD (N), W (N)
      DATA AP / 1, 2,     5,     8,  9,  11/
      DATA AI / 2, 1,3,5, 2,4,5, 3,  2,3   /
C     load the matrix into the AMD workspace
      DO 10 J = 1,N
          PE (J) = AP (J)
          LEN (J) = AP (J+1) - AP (J)
10    CONTINUE
      DO 20 P = 1,NZ
          IW (P) = AI (P)
20    CONTINUE
      PFREE = NZ + 1
C     order the matrix (destroys the copy of A in IW, PE, and LEN)
      CALL AMD (N, PE, IW, LEN, IWLEN, PFREE, NV, NEXT, LAST, HEAD,
     $      ELEN, DEGREE, NCMPA, W)
      DO 60 K = 1, N
          PRINT 50, K, LAST (K)
50        FORMAT ('P (',I2,') = ', I2)
60    CONTINUE
      END
```

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Applic.*, 17(4):886–905, 1996.

[2] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[3] HSL. HSL 2002: A collection of Fortran codes for large scale scientific computation, 2002. `www.cse.clrc.ac.uk/nag/hsl`.