Visualizing Sparse Matrix Computations

Fernando L. Alvarado
The University of Wisconsin
Madison, Wisconsin 53706, USA

Abstract

This paper describes ideas and tools that help with the visualization of sparse matrix computations. The paper uses the Sparse Matrix Manipulation System, an environment for handling sparse matrices of all types in a vy flexible manner via ASCII file interfaces. Two commands in this environment are *ShowMatrix* and *ShowTree*. The first illustrates the pattern of nonzeroes of a sparse matrix. The second illustrates dependencies among matrix elements. Other tools in the package are described, including tools for ordering, factoring and multiplying sparse matrices. This software environment is then used to study the effect of several recent ordering and partitioning algorithms for working with the sparse inverses of L and U. These new algorithms have been proposed as a means of enhancing the parallelism of sparse matrix computations. This paper illustrates in a graphic manner the effect of these algorithms on parallelism and fill-in.

Keywords: Visualization, Sparse Matrices, Large Scale Computation, Software Tools.

1 Introduction

This paper applies a set of matrix visualization tools to the study of W-matrix algorithms (algorithms that work with the inverses of the LDU factors of a matrix A). The work is based on an environment of directly executable commands entitled the *Sparse Matrix Manipulation System* (SMMS) [2,3]. The environment makes use of DOS *filters* and *pipes*. The routines described in this paper have all been developed by the author based on state of the art sparse matrix concepts [8]. The routines are limited only by the capacity of the operating system. Most routines work properly on up to 2000 by 2000 matrices. Users can add to it routines developed in any programming language.

The Sparse Matrix Manipulation System is based on three notions: matrix, permutation vector and partition vector. Each of these is represented as an ASCII file. Communication among all the routines takes place exclusively via one or more of these ASCII files.

The fundamental notion in the Sparse Matrix Manipulation System is the *matrix*. Matrices are specified in `IJvalue` format. The first line contains the dimensions of the matrix. Subsequent lines contain the row index, column index and numeric value of each entry, separated by one or more spaces. A "`0 0 0.0`" line denotes the end of the matrix.

A permutation vector of dimension N is as a vector of integers from 1 to N in arbitrary order. A permutation vector is represented as one data line with a single integer (N, the dimension of the vector), followed by N integers from 1 to N in any order.

The third concept required in the Sparse Matrix Manipulation System is the partition. A partition is a collection of K ordered integers usually denoting positions within the matrix. A partition is specified by a single integer in line 1, denoting the number of partitions, followed by one or more lines with a list of all K partitions.

Table I: Summary of some visualization commands.

Command	Description
`SHOWMAP`	Display matrix topology.
`SHOWVAL`	Display matrix values as a two dimensional array.
`MATSTAT`	Display matrix statistics.
`NORM`	Compute matrix norms.
`LINPACK`	Compute an estimate of the condition number of the matrix using the LinPack estimator.
`SHOWTREE`	Illustrate the factorization path tree.

2 The Sparse Matrix Manipulation System

This section summarizes some of the commands available in the Sparse Matrix Manipulation System.

The most important step in sparse matrix visualization is viewing the nonzero pattern of a matrix. This is accomplished with the `SHOWMAP` command. Figure 1 illustrates the `SHOWMAP` command for a 118 by 118 matrix stored in `IJvalue` format in file `S118.DAT`.

Another visualization command is `SHOWTREE`. This command can be used to visualize relationships among matrix elements by illustrating the factorization path tree of a matrix. Factorization trees require that the matrix have all fill elements already added. The `ADDFILL` command adds missing fills.

In addition, a graphic front end for the SMMS is under development [15]. This front end permits the construction and manipulation of sparse matrices using a mouse-driven graphic interface.

Sparse matrices can be obtained from applications, as output to user developed software, or from libraries of matrices such as the library of test matrices described in [9]. However, in many cases sparse matrices can be generated with the commands as needed. Table II summarizes a few commands available for generating sparse matrices.

Matrix rows and columns can be numbered arbitrarily. Full matrices are usually renumbered to reduce numerical error accumulation. Sparse matrices are usually renumbered to reduce fill-in during factorization. Objectives other than minimum fill-in can be used in the renumbering, such as minimizing elimination tree height. Table III summarizes some permutation commands.

Much of the work required to process matrices is non-numeric. Examples include row and column matrix permutation, discarding of matrix rows and columns, assembly of larger matrices from smaller ones and retention of selected portions of a matrix. Table IV illustrates both numeric and non-numeric commands.

Illustrating the effect of these routines is the heart of this paper. A number of sparse vector and matrix methods based on the inverses of the L and U matrices have been introduced [10]. These methods offer improved performance in parallel environments [4]. Table V describes some of these routines.

Table II: Summary of some matrix generation commands.

Command	Description
`GENRAND`	Generate a random topology sparse matrix of specified size and density.
`GENBAND`	Generate a banded matrix.
`GENGRID`	Generate a matrix for finite element discretization of rectangular grids.
`GENVECT`	Generate a vector.

Table III: Summary of permutation commands.

Command	Description
`TINNEY`	Generate permutation vectors according to the ordering schemes of Tinney [14].
`RANDPERM`	Generate a random permutation vector.
`TRANSVER`	A recursive algorithm to find a transversal [7].
`TARJAN`	Tarjan's algorithm for finding strong blocks.
`GPS`	The Gibbs-Poole-Stockmeyer algorithm.

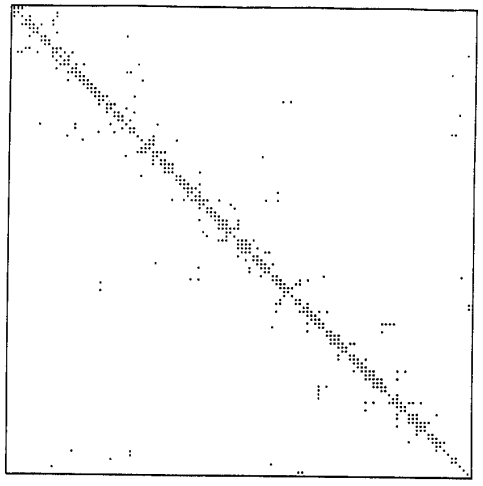

Figure 1: Topology map for a 118 by 118 sparse matrix obtained using SHOWMAP <S118.DAT.

Table IV: Summary of processing commands.	
Command	Description
PERMUTE	Apply a permutation vector to a matrix.
SORTROWS	Sort the matrix rows [1].
ADDFILL	Add fills to a matrix.
TRANSP	Transpose a matrix.
RETAIN	Retain a specified set of rows and columns.
ADJOIN	Construct a larger matrix by adjoining several smaller matrices.
FACTOR	*LDU* factorization of a *perfect elimination topology-symmetric* matrix.
FACTORNS	*LDU* factorization of an *unsymmetric* matrix.
REPSOL	Repeat solution using a factored matrix.
ATIMESB	Multiply two sparse matrices.
SPARSINV	Sparse inverse of a factored sparse matrix [12].
SPARSOL	Sparse vector solutions [13].
FACTUPV	Factor updating using a sparse version of [5].

Table V: Summary of W-matrix commands.	
Command	Description
WMATRIX	Partitioned sparse inverses of L and U (W-factors) [10].
WSOL	Repeat solution using W-factors.
PARTALG	Three partitioning algorithms [4].
GOMEZ	Three ordering algorithms designed to reduce the height of the elimination tree [11].
MLMD	A variation of Tinney Scheme 2 designed to reduce the height of the elimination tree [6].

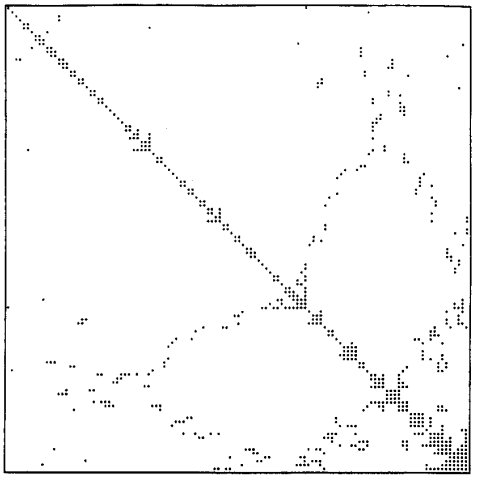

Figure 2: L and U factors of the matrix in Figure 1 ordered according to the minimum degree algorithm (fills added).

3 Visualizing W-matrix methods

The inverse of a sparse matrix A is full (unless A is not strongly connected). If the rows and columns of A are renumbered, A can be factored into a sparse lower triangular matrix L and a sparse upper triangular matrix U. Fill-in inevitably occurs in most problems, regardless of ordering. The inverses of L and U are also sparse (although not as sparse as L and U. Figure 2 illustrates the topology of the L and U factors of the 118 by 118 matrix from Figure 1 after ordering according to the Minimum Degree algorithm and adding all factorization fills. Figure 3 illustrates the *inverses* of L and U. These matrices can be partitioned into products of sub-matrices with no additional fill (Figure 4).

Ordering algorithms other than minimum degree can reduce the number of partitions. Figure 5 illustrates the fills in the matrix from Figure 1 numbered according to the MLMD algorithm. It also illustrates the partitions necessary so no fills occur in W. Figure 6 illustrates the same matrices after partitioning with an algorithm that permits some fill-in in W but reduces the number of partitions.

If W-matrix methods are used, all multiplications for a given partition can take place concurrently. Thus, the number of serial multiplication steps is the number of partitions. Tinney Scheme 2 yields a factorization path graph illustrated in Figure 7. There is no discernible pattern to the partitions. Nineteen levels and 53 partitions result.

The MLMD ordering algorithm results in fewer levels (only 12) and in only seven no-fill partitions (the seventh "partition" is trivial). This is illustrated in Figure 8. Partitioning algorithm PA2 aggregates

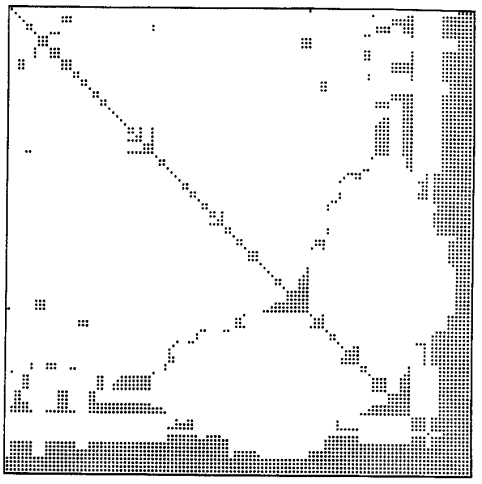

Figure 3: Inverses of the L and U factors (W-factors) in Figure 2. Many fills occur.

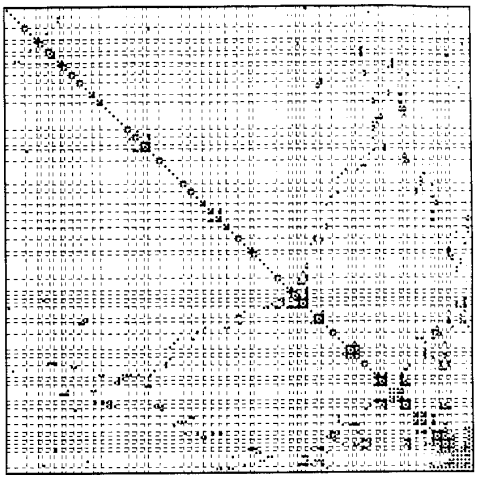

Figure 4: Partitioned inverses of the L and U Figure 2. Many partitions, but no new fills.

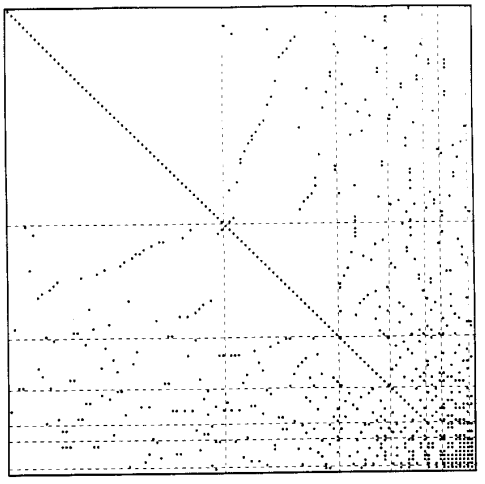

Figure 5: The LU factors for Figure 1 ordered according to the MLMD algorithm. The seven no-fill partitions are also illustrated.

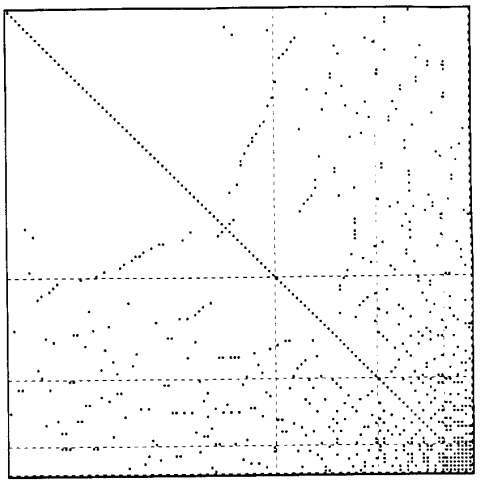

Figure 6: Partitioned inverses of L and U for Figure 5 allowing 10% fills per partition in W. Only partitions are needed.

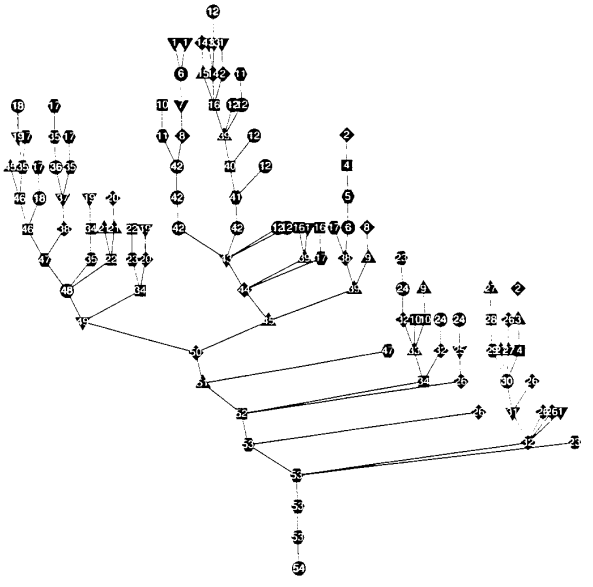

Figure 7: Factorization tree using Tinney Scheme 2. Nineteen levels and 53 partitions result. Partitions are illustrated by numbers and shapes.

elements from different levels into the same partition. Partitions are made up from nodes from different branches in the tree and in some cases from nodes in more than one level of the same branch.

The PA3 ordering algorithm of [4] with the same matrix and the same ordering (MLMD) results in only four partitions. The factorization path tree is *the same* as in Figure 8. However, the grouping of nodes is different. This is illustrated in Figure 9. Here, elements from more levels are grouped into the same partition, at the expense of some additional fills in W.

4 Conclusions

This paper has demonstrated that a toolkit of directly executable commands for the handling of sparse matrices can be of great value in visualizing sparse matrix computations. The almost entirely pictorial illustration of the effects of various W-matrix algorithms and formulations helps understand the behavior and effect.

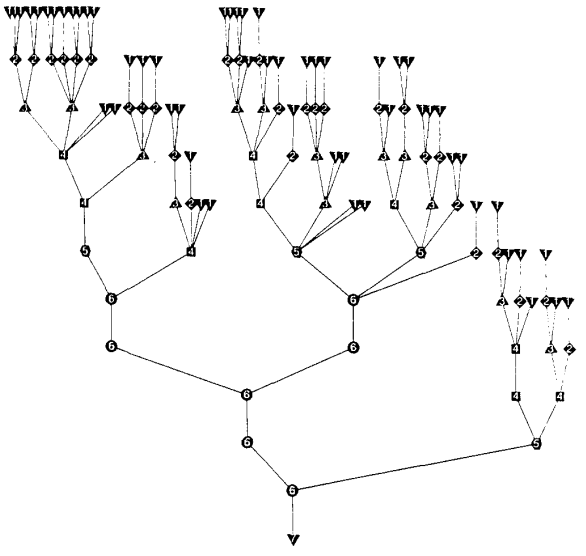

Figure 8: Factorization path tree for matrix ordered according to MLMD algorithm. Twelve levels and seven partitions. Partitioning Algorithm 2 used. Partitions are shown by number and shapes.

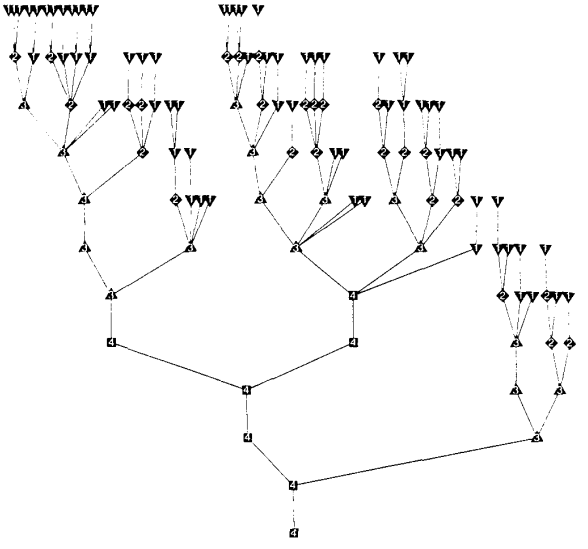

Figure 9: Factorization path tree for matrix ordered using MLMD. Partitioning Algorithm 3. Twelve levels and only four partitions.

Availability

This object code for the Sparse Matrix Manipulations System is available freely for educational and non-commercial research purposes. Copyright notices must be retained in all the routines.

References

[1] F. L. Alvarado, "A Note on Sorting Sparse Matrices," Proc. IEEE, Vol. 67, pp. 1362–1363, Sept. 1979.

[2] F. L. Alvarado, "The Sparse Matrix Manipulation System," The University of Wisconsin, Report ECE-89-1, Jan. 1989.

[3] F. L. Alvarado, "The Sparse Matrix Manipulation System," SIAM Symp. on Sparse Matrices, Gleneden Beach, Or., May 22–24 1989.

[4] F. L. Alvarado, D. Yu and R. Betancourt, "Partitioned Sparse A^{-1} Methods," to appear in IEEE Trans. on Power Systems, May 1990.

[5] J. M. Bennett, "Triangular Factors of Modified Matrices," Numerical Math., Vol. 7, pp. 216–221, 1965.

[6] R. Betancourt, "An Efficient Heuristic Ordering Algorithm for Partial Matrix Refactorization," IEEE Trans. on Power Systems, Vol. 3, pp. 1181–1187, Aug. 1988.

[7] I. S. Duff, "Algorithm 575: Permutations for a Zero-Free Diagonal," ACM Trans. on Math. Software, Vol. 7, pp. 387–390, Sept. 1981.

[8] I. S. Duff, A. M. Erisman and J. K. Reid, "Direct Methods for Sparse Matrices," 1986 Oxford Science Publications, Clarendon Press.

[9] I. S. Duff, R. G. Grimes and J. G. Lewis, "Sparse Matrix Test Problems," ACM Trans. on Math. Software, Vol. 15, pp. 1–14, March 1989.

[10] M. K. Enns, W. F. Tinney and F. L. Alvarado, "Sparse Matrix Inverse Factors," IEEE/PES 1988 Summer Meeting, July 19-24, Paper 88 SM 728-8.

[11] A. Gómez and L. G. Franquelo, "An Efficient Ordering Algorithm to Improve Sparse Vector Methods," IEEE Transactions on Power Systems, Vol. 3, No. 4, pp. 1538–1544, November 1988.

[12] K. Takahashi, J. Fagan and M. Chen, "Formulation of a Sparse Bus Impedance Matrix and its Application in Short Circuit Studies," Proc. PICA Conf., pp. 63–69. Minneapolis, Minn., July 1973.

[13] W. F. Tinney, V. Brandwajn and S. M. Chan, "Sparse Vector Methods," IEEE Trans. on Power Apparatus and Systems, Vol. 104, pp. 295–301, Feb. 1985.

[14] W. F. Tinney and J. W. Walker, "Direct Solution of Sparse Network Equations by Optimally Ordered Triangular Factorization," Proc. IEEE, Vol. 55, pp. 1801–1809, Nov. 1967.

[15] D. Yu, S. T. Chen and F. L. Alvarado, "An Interactive Graphic Interface to Illustrate Sparse Matrix/Vector Methods," IEEE/PES 1990 Winter Meeting, Atlanta, GA, February 4–9, 1990.