

# On the Perfect Elimination Ordering of Nonsymmetric Matrices

Tzu-Yi Chen\*

Dept. of Math and Computer Science  
Pomona College  
Claremont, CA 91711

Melissa Egan†

Dept. of Math and Computer Science  
Pomona College  
Claremont, CA 91711

## Abstract

A direct method for solving a system of equations  $Ax = b$  first factors the matrix  $A$  into the product of lower and upper triangular matrices and then computes the solution vector  $x$ . When  $A$  is large and sparse, reordering  $A$  prior to computing its factor may be necessary to keep memory requirements reasonable. If a matrix can be reordered so that the space required for its factors is no greater than that needed for the original matrix, we say the matrix is *perfect elimination*. In this paper we ask how close matrices from actual applications come to being perfect elimination. Tests run on matrices from various application areas find several matrices that are perfect elimination, and several others that are nearly so.

**Keywords:** sparse nonsymmetric matrices, perfect elimination ordering, direct methods

## 1 Introduction

Applications ranging from computational fluid dynamics to circuit simulation depend on methods for solving systems of linear equations  $Ax = b$ . The input to the method should be the  $n \times n$  nonsingular matrix  $A$  and the vector  $b$ ; the output should be the solution vector  $x$ . Direct methods, which are considered more robust than iterative methods, begin by computing the LU factorization of  $A$  (ie, lower and upper triangular matrices  $L$  and  $U$  such that  $LU = A$ ). They then find  $x$  by efficiently solving two triangular systems of equations ( $Ly = b$  and  $Ux = y$ ).

In practice, the matrix  $A$  is often both large and *sparse*. A sparse matrix is one where enough entries have value 0.0 that significant space can be saved by storing only the values of the nonzero entries and information about their locations in the matrix. Unfortunately, even when  $A$  is sparse, its  $L$  and  $U$  factors may be dense. For example, consider the matrix  $A$  in Figure 1. The  $n \times n$  matrix has  $3n - 2$  nonzeros (ie,  $\text{nnz}(A) = 3n - 2$ ), yet  $\text{nnz}(L + U) = n^2$ .

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \ddots & & \\ \times & & \times & \\ \times & & & \times \end{bmatrix} = \begin{bmatrix} \times & & & \\ \vdots & \ddots & & \\ \times & \times & \times & \\ \times & \times & \times & \times \end{bmatrix} \times \begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \ddots & \vdots \\ & & & \times \end{bmatrix}$$

Figure 1: A sparse matrix  $A$  with dense LU factors. Elements marked by a  $\times$  have nonzero values.

We refer to nonzero elements in  $L + U$  that are not present in  $A$  as *fill*. Excessive fill can render a direct method impractical if the user lacks the computer memory necessary to store the factors. Fortunately,

---

\*This author qualifies as a new investigator. She received her PhD in computer science from UC Berkeley in December, 2001.

†This author qualifies as a new investigator. She graduated from Pomona College in 2003 and will begin graduate study at the Naval Postgraduate School in late March, 2004.

permuting the rows and columns of  $A$  before computing  $L$  and  $U$  can greatly reduce the fill. For example, consider Figure 2, which shows the effect of reverse ordering the rows and columns of the matrix in Figure 1. The factorization now introduces no fill (ie,  $\text{nnz}(A) = \text{nnz}(L + U)$ ). If a given matrix can be permuted to fully eliminate fill, we call it a *perfect elimination* matrix; the associated permutation is called a *perfect elimination ordering*.<sup>1</sup>

$$\begin{bmatrix} \times & & & \times \\ & \times & & \times \\ & & \ddots & \times \\ \times & \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & & & \\ & \times & & \\ & & \ddots & \\ \times & \times & \times & \times \end{bmatrix} \times \begin{bmatrix} \times & & & \times \\ & \times & & \times \\ & & \ddots & \times \\ & & & \times \end{bmatrix}$$

Figure 2: A matrix  $A$  whose LU factorization introduces no fill. This is the same matrix as that in Figure 1, but with the rows and columns in reverse order.

Given an arbitrary matrix, is it perfect elimination? If the matrix is structurally symmetric (ie, the nonzero pattern of  $A$  is the same as that of  $A^T$ ), and the rows and columns must be permuted in the same way, there are linear time algorithms for determining whether it is perfect elimination (eg, [9]). If the rows and columns may be permuted differently, as one may want to do with a nonsymmetric matrix, there is an  $O(n^3)$  algorithm [6].

Are there nonsymmetric matrices from actual applications that are perfect elimination? The surprising answer is “yes.” In this paper we begin with an overview of algorithms for determining whether a nonsymmetric matrix is perfect elimination. We then describe our implementation of the algorithm in [6] which, to the best of our knowledge, has never before been implemented, let alone tested on real matrices. Finally, in running the algorithm on over a hundred matrices from actual applications, we find several matrices that are perfect elimination. Furthermore, of the ones that are not, many have a large number of entries that can be eliminated without introducing fill.

## 2 Algorithms

The known algorithms for determining whether a nonsymmetric matrix is perfect elimination rely on Theorem 1 from [8]. Although Theorem 1 is stated in terms of bipartite graphs, references such as [7] discuss the natural correspondence between bipartite graphs and  $(0, 1)$  matrices (ie, matrices whose nonzero entries all have value 1). For the rest of this paper we use  $A_1$  to refer to the matrix with the same nonzero structure as  $A$ , but with nonzero values set to 1.

**Theorem 1** *If  $e$  is a bisimplicial edge of a perfect elimination bipartite graph  $G$ , then  $G - e$  is also a perfect elimination bipartite graph.*

The theorem, in terms of matrices, says that if an entry of a perfect elimination matrix can be used as a pivot without introducing fill, then the rest of the matrix can still be permuted in a way that generates no fill. The  $a_{ij}$  element in the matrix  $A$  can be used as a pivot without introducing fill if  $a_{ij} \neq 0$  and if, in  $A_1$ , every row with an element in column  $j$  majorizes row  $i$ . Informally, in a  $(0, 1)$  matrix, row  $i_1$  majorizes row  $i_2$  if it has a nonzero element in at least every column that  $i_2$  does.

### 2.1 An $O(n^5)$ algorithm

The authors of [8] note Theorem 1 immediately suggests an algorithm for determining whether a matrix has a perfect elimination ordering: repeatedly search for elements which create no fill when used as pivots, then

<sup>1</sup>For most of this paper we are concerned only with the nonzero structure of the matrix, and not with the values of the nonzero entries. Therefore, we assume there is no numerical cancellation, which can introduce unexpected 0 elements in the factors.

```

PerfectElimination( $A$ )
1    $M = A_1$ 
2   isPerfectElimination  $\leftarrow$  true
3   compute  $Q = MM^T$ 
4   for  $j \leftarrow 1$  to  $n$  do
5        $s_j \leftarrow \sum_{i=1}^n m_{ij}$ 
6   while there exists an  $s_j \neq 0$  and isPerfectElimination do
7       for  $i \leftarrow 1$  to  $n$  do
8            $l_i \leftarrow$  the number of entries in row  $i$  of  $Q$  which are equal to  $q_{ii}$ 
9       if there is a nonzero entry  $m_{ij}$  such that  $s_j = l_i$  then
10          Compute the matrix  $D = (d_{kl})$  where  $d_{kl} = m_{kj} * m_{lj}$ ;
11           $Q \leftarrow (Q - D)^{ii}$  (*)
12          for  $k \leftarrow 1$  to  $n$  do
13               $s_k \leftarrow s_k - m_{ik}$ 
14           $s_j \leftarrow 0$ 
15           $M \leftarrow M^{ij}$  (**)
16      else isPerfectElimination  $\leftarrow$  false

```

Figure 3: Pseudocode from [6] describing an  $O(n^3)$  algorithm that determines whether a matrix is perfect elimination. (\*)  $Q$  is now equal to  $(M^{ij})(M^{ij})^T$  (\*\*) This line is, incorrectly, omitted in [6].

delete them by zeroing out the row and column containing that element. The matrix is perfect elimination if and only if this can be repeated until the matrix is empty.

The running time of this algorithm is  $O(n^5)$ , where  $n$  is the dimension of the matrix. In each of  $n$  iterations up to  $n^2$  elements are checked for eliminability; each check requires looking at up to  $n^2$  elements.

## 2.2 An $O(n^3)$ algorithm

In [6], Goh and Rotem prove the following lemma and use it to improve on the algorithm in [8].

**Lemma 1** *Let  $M = (m_{ij})$  be an  $n \times n$   $(0,1)$  matrix representing a bipartite graph  $G = (X \cup Y, E)$ . Let  $l_i$  be the number of rows in  $M$  that majorize row  $i$  and  $s_j$  the sum of entries in column  $j$  of  $M$ . Then  $m_{ij} = 1$  and  $l_i = s_j$  if and only if the edge  $x_i y_j$  is a bisimplicial edge of  $G$ .*

In other words, if  $M = A_1$ , using the nonzero element  $a_{ij}$  as a pivot creates no fill if and only if  $s_j$ , the number of nonzeros in column  $j$ , equals  $l_i$ , the number of rows that majorize row  $i$  in  $M$ . The reasoning is as follows: clearly  $s_j \geq l_i$ , since  $m_{ij}$  is nonzero and therefore any row that majorizes row  $i$  must have a nonzero value in column  $j$ . Furthermore,  $s_j \leq l_i$  since if  $s_j > l_i$ , then there is some row  $k$  such that  $m_{kj} \neq 0$ , yet for which  $m_{kr} = 0$  when  $m_{ir} \neq 0$ . This would mean using  $m_{ij}$  as a pivot would create fill by placing a nonzero value in  $m_{kr}$ . This shows that if  $m_{ij}$  creates no fill as a pivot, then  $s_j = l_i$ .

In Figure 3 we give an adapted version of the pseudocode in [6]. The initial values of  $s_j$  are easily computed. The algorithm calculates the value of  $l_i$  for all  $i$  by first computing  $Q = A_1 A_1^T$ . Now  $q_{ij}$  is the number of columns in  $A$  with a nonzero entry in both row  $i$  and row  $j$ . This means  $q_{ii}$  is the number of nonzero entries in row  $i$ , so  $q_{ij} = q_{ii}$  if and only if row  $j$  majorizes row  $i$ . Hence  $l_i$  is the number of elements in row  $i$  of  $Q$  that are equal to  $q_{ii}$ . Note that  $M^{ij}$  denotes the matrix  $M$  with all entries in row  $i$  and column  $j$  set to 0.

In the worst case this is an  $O(n^3)$  algorithm. The matrix  $Q$  is computed only once, and is then updated in each iteration. The  $s$ -values and matrix  $M$  must also be updated in each iteration, and the  $l$ -values must be recalculated as each entry is deleted. The updates take  $O(n^2)$  time and are repeated up to  $n$  times, making the total running time of the algorithm  $O(n^3)$ .

### 3 Implementation

Now we turn to our implementation of the algorithm in [6]. We begin by explaining how the sparse matrices given as input are stored, and then summarize our implementation and subsequent optimization decisions.

#### 3.1 Storing sparse matrices

Given a sparse  $n \times n$  matrix, users tend not to store all the values of all  $n^2$  elements. Instead they store the values of the nonzero elements, together with information about where they are located in the matrix. The simplest storage format might be the *triplet* format, in which an entry of the form  $(i, j, v)$  means that  $a_{ij} = v$ .

In practice more compact representations are used. A standard format is the column-compressed format, sometimes referred to as the Harwell-Boeing format. Here a matrix is represented by three arrays. The first is `nzval`, which stores the nonzero values in the matrix, beginning with those in the first column, then those in the second column, and so on. The second is `rowind`, where `rowind[i]` contains the row index of the element whose value is stored in `nzval[i]`. The last is `colptr`, where `colptr[j]` gives the index of the first element in `rowind` that is in column  $j$ . Note that `colptr` is of size `n+1`, and the value of the last element must be the number of nonzero elements in the matrix. As the algorithm assumes the input is a  $(0, 1)$  matrix, we only use the `rowind` and `colptr` arrays.

For more information on triplet, column-compressed, and other formats for storing sparse matrices, see [1].

#### 3.2 Implementation details and optimizations

We implemented the algorithm in C, making use of the I/O functions for matrices stored in column-compressed format provided at [2]. The process of calculating  $Q$  requires storage of size  $O(n + nnz(A) + nnz(Q))$  and takes time  $O(n^2 + n \cdot nnz(A))$ , where  $n$  is the dimension of  $A$ ,  $nnz(A)$  is the number of nonzero elements in  $A$ , and  $nnz(Q)$  is the number of nonzero elements in  $Q = AA^T$ . Once  $Q$  is created, it is updated in place. Storing the  $s$ - and  $l$ - values requires two integer arrays, each of size  $n$ .

With a few exceptions, noted here, our implementation follows the pseudocode in [6], given in Figure 3. First, when calculating the  $l$ -values in line 8, we evaluate the columns of  $Q$  rather than the rows. Because  $MM^T = (MM^T)^T$ , we know  $Q$  is symmetric and so the operations are equivalent. In addition, because our matrices are stored in column-compressed format, it is more efficient to retrieve columns than rows. Note that we do not recalculate  $l$ -values for rows which have already been removed from  $Q$ .

Second, when updating  $Q$  in lines 10-11, we do not compute  $D$  and then subtract it from  $Q$ . Our code updates only those columns of  $Q$  with entries that have changed with the removal of the last pivot. More specifically, if the last element removed from  $M$  was in column  $j$ , then for each nonzero element  $m_{kj}$ , column  $k$  of  $Q$  must be updated. The nonzero elements in any given column  $c$  of  $M$  are those elements with indices between `colptr[c]` and `colptr[c+1]-1`.

Finding an eliminable element and making the updates necessary to remove it takes  $O(n + nnz(A) + nnz(Q))$  time, and is repeated up to  $n$  times. Thus the overall algorithm takes  $O(n^2 + n \cdot nnz(A) + n \cdot nnz(Q))$  time.

### 4 Results and Analysis

After testing the code on small examples for which it was known whether or not the matrix was perfect elimination, we moved on to a test suite consisting of 180 matrices from actual applications. The matrices came from both the University of Florida sparse matrix collection [3] and the Matrix Market [2]. All tests were run on a 2 GHz Intel Xeon processor with 2 GBytes of memory.

In Figure 4 we show the number of entries, as a percentage of the dimension  $n$ , that can be eliminated in each matrix without introducing fill. If the percentage is 100, then the matrix is perfect elimination. Notice that 19 of the 180 matrices (over 10%) are perfect elimination. For these matrices, the perfect elimination

ordering is not simply the original ordering of the matrix, nor is the perfect elimination ordering discovered by popular nonsymmetric fill-reducing heuristics such as COLAMD [4]. Figure 4 also shows that almost half of the matrices contain 40%, or more, eliminable entries.

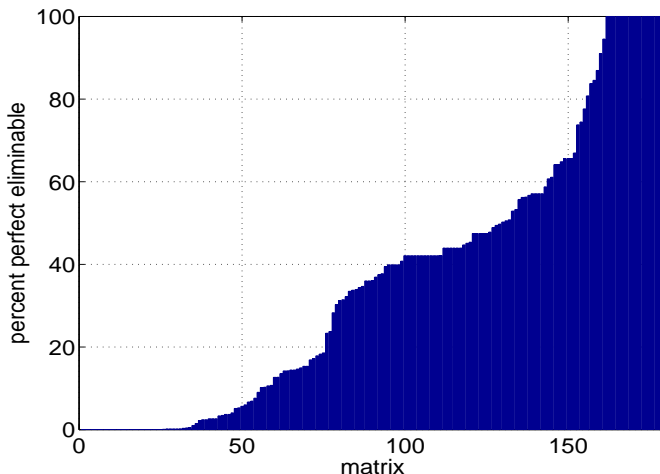


Figure 4: The number of entries, expressed as a percentage of the dimension, that can be consecutively eliminated without introducing fill.

Although so far we have concentrated on the nonzero structure of the matrices and not on the numerical values of their entries, we now change course. When factoring a nonsymmetric matrix, a technique known as *pivoting* is often needed for numerical stability. Pivoting, discussed in most books on linear algebra, simply refers to moving large entries of a matrix to the diagonal during the factorization. Not pivoting can result in tiny entries on the diagonal, which is sometimes referred to as *pivot growth*. Unfortunately, while necessary for stability, pivoting can destroy a precomputed perfect elimination ordering.

Our second set of tests looked at the pivot growth when using a perfect elimination ordering, and then looked at the amount of fill created when pivoting was used in combination with a perfect elimination ordering. More specifically, the matrices that were found to be perfect elimination were ordered using their perfect elimination ordering, and then factored using the package SuperLU [5]. We computed the factorizations both with and without partial pivoting and looked at both the pivot growth and at the number of nonzeros in the  $L$  and  $U$  factors.<sup>2</sup>

The results are shown in Table 1. Note that when there were several perfect elimination matrices that belonged to the same family (eg, olm100, olm500, olm1000, and olm5000), we included only one in Table 1. As expected for these perfect elimination matrices, without partial pivoting the number of nonzeros in  $L+U$  was found to equal the number of nonzeros in  $A$ ; however, the pivot growth was occasionally poor. With partial pivoting the pivot growth is good (near 1.0), though there is sometimes increased fill. Two facts stand out. First, except for the olm5000 matrix, the pivot growth is not terrible even without pivoting. Second, introducing partial pivoting generally does not increase the fill by very much, if at all.

A table of results, including the names of the matrices and the percent to which each is perfectly eliminable, can be found at <http://www.cs.pomona.edu/~tzuyi/PerfectElimination>.

## 5 Conclusions

In this paper we presented what we believe to be the first implementation of an algorithm for determining whether nonsymmetric matrices are perfect elimination. Our extensive tests on matrices from various actual

<sup>2</sup>For SuperLU to give a precise number for  $nnz(L+U)$ , set the relaxation parameter `relax` in the `sp_ienv.c` file to 1.

matrix	n(A)	nnz(A)	without pivoting		with partial pivoting	
			nnz(L+U)	recip. piv. growth	nnz(L+U)	recip. piv. growth
add32	4960	23884	23884	1.0	23884	1.0
bp_0	822	3276	3276	1.0	3276	1.0
memplus	17758	126150	126150	1.0	126678	1.0
mhd416b	416	2312	2312	1.0	2524	.881
olm5000	5000	19996	19996	.000415	34954	.6
shl_400	663	1712	1713 (*)	1.0	1713	1.0
str_0	363	2454	2454	1.0	2454	1.0
tols4000	4000	8784	8784	.999	8784	.999

Table 1: The effect of pivoting on (reciprocal) pivot growth and fill for matrices with perfect elimination orderings. (\*) This is the number of nonzeros in  $L + U$  as reported by SuperLU [5]. Tests in Matlab show that there are, in fact, only 1712 nonzeros in the factors.

applications shows that a surprising number are either perfect elimination, or are nearly so.

While this is an interesting result in and of itself, we are currently working on adapting our code to be a general heuristic for reducing fill for nonsymmetric matrices. Very preliminary results suggest that it may be difficult to outperform a heuristic such as COLAMD [4].

## References

- [1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. v. d. Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.
- [2] R. Barrett, R. Boisvert, J. J. Dongarra, R. Lipman, B. Miller, R. Pozo, and K. Remington. Matrix Market. <http://math.nist.gov/MatrixMarket>
- [3] T. Davis. University of Florida sparse matrix collection. NA Digest, v.92, n.42, Oct. 16, 1994 and NA Digest, v.96, n.28, Jul. 23, 1996, and NA Digest, v.97, n.23, Jun. 7, 1997. <http://www.cise.ufl.edu/research/sparse/matrices/>
- [4] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. A column approximate minimum degree ordering algorithm. Technical Report TR-00-005, Department of Computer and Information Science and Engineering, University of Florida, October 2000.
- [5] J. W. Demmel, J. R. Gilbert, and X. S. Li. *SuperLU users' guide*, September 1999. <http://www.nersc.gov/~xiaoye/SuperLU/>
- [6] L. Goh and D. Rotem. Recognition of perfect elimination bipartite graphs. *Inform. Process. Lett.*, 15(4):179–182, 1982.
- [7] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. computer science and applied mathematics. Academic Press, New York, 1980.
- [8] M. C. Golumbic and C. F. Goss. Perfect elimination and chordal bipartite graphs. *Journal of Graph Theory*, 2(2):155–163, 1978.
- [9] B. S. Panda. New linear time algorithms for generating perfect elimination orderings of chordal graphs. *Inform. Process. Lett.*, 58:111–115, 1996.