Nested-Dissection Orderings for Sparse LU with Partial Pivoting

Igor Brainman[1] and Sivan Toledo[1]

School of Mathematical Sciences, Tel-Aviv University
Tel-Aviv 69978, ISRAEL
Email: sivan@math.tau.ac.il
Home page: http://www.math.tau.ac.il/~sivan

Abstract. We describe the implementation and performance of a novel fill-minimization ordering technique for sparse LU factorization with partial pivoting. The technique was proposed by Gilbert and Schreiber in 1980 but never implemented and tested. Like other techniques for ordering sparse matrices for LU with partial pivoting, our new method preorders the columns of the matrix (the row permutation is chosen by the pivoting sequence during the numerical factorization). Also like other methods, the column permutation Q that we select is a permutation that minimizes the fill in the Cholesky factor of $Q^T A^T A Q$. Unlike existing column-ordering techniques, which all rely on minimum-degree heuristics, our new method is based on a nested-dissection ordering of $A^T A$. Our algorithm, however, never computes a representation of $A^T A$, which can be expensive. We only work with a representation of A itself. Our experiments demonstrate that the method is efficient and that it can reduce fill significanly relative to the best existing methods. The method reduces the LU running time on some very large matrices (tens of millions of nonzeros in the factors) by more than a factor of 2.

1 Introduction

Reordering the columns of sparse nonsymmetric matrices can significantly reduce fill in sparse LU factorizations with partial pivoting. Reducing fill in a factorization reduces the amount of memory required to store the factors, the amount of work in the factorization, and the amount of work in subsequent triangular solves. Symmetric positive definite matrices, which can be factored without pivoting, are normally reordered to reduce fill by applying the same permutation to both the rows and columns of the matrix. When partial pivoting is required for maintaining numerical stability, however, pre-permuting the rows is meaningless, since the rows are exchanged again during the factorization. Therefore, we normally preorder the columns and let numerical consideration dictate the row ordering. Since columns are reordered before the row permutation is known, we need to order the columns such that fill is minimized no matter how rows are exchanged. (Some nonsymmetric factorization codes that employ pivoting, such as UMFPACK/MA38 [2, 3], determine the column permutation during the numerical factorization; such codes do not preorder columns so the technique in this paper does not apply to them.)

A result by George and Ng [6] suggests one effective way to preorder the columns to reduce fill. They have shown that the fill of the LU factors of PA is essentially contained in the fill of the Cholesky factor of A^TA for every row permutation P. (P is a permutation matrix that permutes the rows of A and represents the actions of partial pivoting.) Gilbert [8] later showed that this upper bound on the fill of the LU factors is not too loose, in the sense that for a large class of matrices, for every fill element in the Cholesky factor of A^TA there is a pivoting sequence P that causes the element to fill in the LU factors of A. Thus, nonsymmetric direct sparse solvers often preorder the columns of A using a permutation Q that minimizes fill in the Cholesky factor of Q^TA^TAQ.

The main challenge in column-ordering algorithms is to find a fill-minimizing permutation without computing A^TA or even its nonzero structure. While computing the nonzero structure of A^TA allows us to use existing symmetric ordering algorithms and codes, it may be grossly inefficient. For example, when an n-by-n matrix A has nonzeros only in the first row and along the main diagonal, computing A^TA takes $\Omega(n^2)$ work, but factoring it takes only $O(n)$ work.

This challenge has been met for the class of reordering algorithms based on the minimum-degree heuristic. Modern implementations of minimum-degree heuristics use a *clique-cover* to represent the graph G_A of the matrix[1] A (see [5]). A clique cover represents the edges of the graph (the nonzeros in the matrix) as a union of cliques, or complete subgraphs. The clique-cover representation allows us to simulate the elimination process with a data structure that only shrinks and never grows. There are two ways to initialize the clique-cover representation of G_{A^TA} directly from the structure of A. Both ways create a data structure whose size is proportional to the number of nonzeros in A, not the number of nonzeros in A^TA. From then on, the data structure only shrinks, so it remains small even if A^TA is relatively dense. In other words, finding a minimum-degree column ordering for A requires about the same amount of work and memory as finding a symmetric ordering for $A^T + A$, the symmetric completion of A.

Nested-dissection ordering methods were proposed in the early 1970's and have been known since then to be theoretically superior to minimum-degree methods for important classes of sparse symmetric definite matrices. Only in the last few years, however, have nested-dissection methods been shown experimentally to be more effective than minimum-degree methods.

In 1980 Gilbert and Schreiber proposed a method for ordering G_{A^TA} using nested-dissection heuristics, without ever forming A^TA [7, 9]. Their method uses *wide separators*, a term that they coined. They have never implemented or tested their proposed method.

The main contribution of this paper is an implementation and an experimental evaluation of the wide-separator ordering method, along with a new presentation of the theory of wide separators.

[1] The graph $G_A = (V, E)$ of an n-by-n matrix A has a vertex set $v = \{1, 2, \ldots, n\}$ and an edge set $E = \{(i, j) | a_{ij} \neq 0\}$. We ignore numerical cancellations in this paper.

Modern symmetric ordering methods generally work as follows:

1. The methods find a small vertex separator that separates the graph G into two subgraphs with roughly the same size.
2. Each subgraph is dissected recursively, until each subgraph is fairly small (typically several hundred vertices).
3. The separators are used to impose a coarse ordering. The vertices in the top-level separator are ordered last, the vertices in the second-to-top level come before them, and so on. The vertices in the small subgraphs that are not dissected any further appear first in the ordering. The ordering within each separator and the ordering within each subgraph has not yet been determined.
4. A minimum-degree algorithm computes the final ordering, subject to the coarse ordering constraints.

While there are many variants, most codes use this overall framework.

Our methods apply the same framework to the graph of $A^T A$, but without computing it. We find separators in $A^T A$ by finding *wide separators* in $A^T + A$. We find a wide separator by finding a conventional vertex separator and widening it by adding to it all the vertices that are adjacent to the separator in one of the subgraphs. Such a wide separator corresponds to a vertex separator in $A^T A$. Just like symmetric methods, our methods recursively dissect the graph, but using wide separators. When the remaining subgraphs are sufficiently small, we compute the final ordering using a constrained column-minimum-degree algorithm. We use existing techniques to produce a minimum-degree ordering of $A^T A$ without computing $G_{A^T A}$ (either the row-clique method or the augmented-matrix method).

Experimental results show that our method can reduce the work in the LU factorization by up to a factor of 3 compared to state-of-the-art column-ordering codes. The running times of our method are higher than the running-times of strict minimum-degree codes, such as COLAMD [10], but they are low enough to easily justify using the new method. On many matrices, including large ones, our method significanly reduces the work compared to all the existing column ordering methods. On some matrices, however, constraining the ordering using wide-separators increase fill rather than reduce it.

The rest of the paper is organized as follows. Section 2 presents the theory of wide separators and algorithms for finding them. Our experimental results are presented in Section 3. We discuss our conclusions from this research in Section 4.

2 Wide Separators: Theory and Algorithms

Our column-ordering methods find separators in $G_{A^T A}$ by finding a so-called *wide separator* in G_{A^T+A}. We work with the graph of $A^T + A$ and not with G_A for two reasons. First, this simplifies the definitions and proofs. Second, to the best of our knowledge all existing vertex-separator codes work with undirected graphs, so there is no point in developping the theory for the directed graph G_A.

A vertex subset $S \subseteq V$ of an undirected graph $G = (V, E)$ is a *separator* if the removal of S and its incident edges breaks the graph into two components $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, such that any path between $i \in V_1$ and $j \in V_2$ passes through at least one vertex in S. A vertex set is a *wide separator* if every path between $i \in V_1$ and $j \in V_2$ passes through a sequence of two vertices in S (one after the other along the path).

Our first task is to show that every wide separator in G_{A^T+A} is a separator in G_{A^TA}. (proofs are omitted from this abstract due to lack of space)

Theorem 1. A wide separator in G_{A^T+A} is a separator in G_{A^TA}.

The converse is not always true. There are matrices with separators in G_{A^TA} that do not correspond to wide separators in A^T+A. The converse of the theorem is true, however, when there are no zeros on the main diagonal of A:

Theorem 2. If there are no zeros on the diagonal of A, then a separator in G_{A^TA} is a wide separator in G_{A^T+A}.

Given a code that finds conventional separators in an undirected graph, finding wide separators is easy. The separator and its neighbors in either G_1 or G_2 form a wide separator:

Lemma 1. *Let S be a separator in an undirected graph G. The sets $S_1 = S \cup \{i | i \in V_1, (i,j) \in E$ for some $j \in S\}$ and $S_2 = S \cup \{i | i \in V_2, (i,j) \in E$ for some $j \in S\}$ are wide separators in G.*

The proof of the theorem is trivial. The sizes of S_1 and S_2 are bounded by $d|S|$, where d is the maximum degree of vertices in S. Given S, it is easy to enumerate S_1 and S_2 in time $O(d|S|)$. This running time is typically insignificant compared to the time it takes to find S.

Which one of the two candidate wide separators should we choose? A wide separator that is small and that dissects the graph evenly reduces fill in the Cholesky factor of A^TA, and hence in the LU factors of A. The two criteria are usually contradictory. Over the years it has been determined the the best strategy is to choose a separator that is as small as possible, as long as the ratio of the number of vertices in G_1 and G_2 does not exceed 2 or so.

The following method, therefore, is a reasonable way to find a wide separator: Select the smallest of S_1 and S_2, unless the smaller wide separator unbalances the separated subgraphs (so that one is more than twice as large as the other) but the larger does not. Our code, however, is currently more naive and always choose the smaller wide separator.

3 Experimental Results

3.1 Experimental Setup

The experiments that this section describe test the effectiveness and performance of several column-ordering codes. We have tested our new codes, which implement nested-dissection-based orderings, as well as several existing ordering codes.

Our codes build a hierarchy of wide separators and then use the separators to constrain a minimum-degree algorithm. We obtain the wide separators by widening separators in G_{A^T+A} that SPOOLES [1] finds. SPOOLES is a new library of sparse ordering and factorization codes that is being developped by Cleve Ashcraft and others. Our codes then invoke a column-mininum-degree code to produce the final ordering. One minimum-degree code that we use is SPOOLES's multi-stage-minimum-degree (MSMD) algorithm, which we run on the augmented matrix. The other minimum-degree code that we used is a version of COLAMD [10] that we modified to respect the constraints imposed by the separators.

The existing minimum-degree codes that we have tested include COLAMD, SPOOLES's MSMD (operating on the augmented matrix with no separator constraints), and COLMMD, a column minimum-degree code, originally written by Joseph W.-H. Liu and distributed with SuperLU.

We use the following acronims to refer to the ordering methods: MSMD refers to SPOOLES' minimum-degree code operating on the augmented matrix without constraints, WS+MSMD refers to the same minimum-degree code but constrained to respect wide separators, and similarly for COLAMD and WS+COLAMD.

In one set of experiments we first reduced the matrices to *block triangular form* (see [12]) applied the ordering and factorization to the diagonal blocks in the reduced form.

We always factor the reordered matrix using SuperLU [4, 11], a state-of-the-art sparse-LU-with-partial-pivoting code. SuperLU uses the BLAS; we used the standard Fortran BLAS for the experiments. We plan to use a higher-performance implementation of the BLAS for the final version of the paper.

We conducted the experiments on a 500MHz dual Pentium III computer with 1 GByte of main memory running Linux. This machine has two processors, but our code only uses one processor.

We tested the ordering methods on a set of nonsymmetric sparse matrices from Tim Davis's sparse matrix collection[2]. We used all the nonsymmetric matrices in Davis's collection that were not too small (less than 0.1 second factorization time with one of the ordering methods) and that did not require more than 1Gbytes to factor. The matrices are listed in Table 1. For further details about the matrices, see Davis's web site (the final version of this paper will include a table listing the order and number of nonzeros for each matrix; the table is omitted from this abstract due to lack of space).

3.2 Results and Analysis

Table 1 summarizes the results of our experiments. The table shows experiments without reduction to block triangular form.

[2] `http://www.cise.ufl.edu/~davis/sparse/`

Columns 2–9 in the table show that wide-separator ordering techniques are effective. Wide separator (WS) orderings are the most effective ordering methods, in terms of work in the factorization, on 23 out of the 41 test matrices. WS orderings are the most effective on 9 out of the 10 largest matrices (largest in terms of work in the factorization). On the single matrix out of the 10 largest where a WS ordering was not the best, it required only 7% more flops to factor.

The reduction in work due to wide separators is often significant. On the larget matrix in our test suite, `li`, wide separators reduce factorization work by almost a factor of 2. The reduction compared to the unconstrained MD methods is also highly significant on `raefsky3`, `epb3`, and `graham1`.

When WS orderings do poorly compared to MD methods, however, they sometimes do significantly poorer. On `ex40`, for example, using wide separators requires 2.66 times the number of flops that COLAMD alone requires. The slowdowns on some of the `lhr` and `bayer` matrices are even more dramatic, but reduction to block triangular form often resolves these problems.

On `lhr14c`, for example, reduction to block triangular form prior to the ordering and factorization reduced the ordering time by more than a factor of 10 and reduced the number of nonzeros in MSMD+WS from $2.1e9$ to $8.2e7$ (and to $4.5e7$ for MSMD alone). These experiments are not reported here in detail because we conducted them too late. The complete results will appear in the final version of the paper.

As columns 7–9 in the table show, reducing flop counts generally translates into reducing the running time of the factorization algorithm and reducing the size of the LU factors. The detailed comparisons between ordering methods other than COLAMD and WS+COLAMD are similar and are omitted from the table. Hence, our remarks concerning the flop counts above also apply to the running time of the factorization code and the amount of memory required to carry out the factorization and to store the factors.

Wide-separator orderings are more expensive to compute than strict minimum-degree orderings, but the extra cost is typically small compared to the subsequent factorization time. Column 10 in the table shows the cost of ordering relative to the cost of the factorization. The table shows that a few matrices take longer (sometimes much longer) to order than to factor. This happens to matrices that arise in chemical engineering (the `bayer` matrices and the `lhr` matrices). We hope to resolve this issue using reduction to block tridiagonal form. Another point that emerges from the table is that on small matrices, wide-separator orderings are expensive to compute relative to the cost of the factorization.

4 Conclusions And Future Work

Our main conclusion from this research is that hybrid wide-separator/minimum-degree column orderings are effective and inexpensive to compute. They often reduce substantially the amount of time and storage required to factor a sparse matrix with partial pivoting, compared to minimum-degree orderings such as COLAMD and COLMMD. They are more expensive to compute than minimum-

Table 1. A comparison of wide-separator and mimimum-degree column orderings. Columns 2–6 show the number of floating-point operations (flops) required to factor the test matrices using 5 different ordering methods. The flop counts for the most efficient method (or methods) are printed in bold. Columns 7–9 show the effectiveness of WS+COLAMD relative to that of COLAMD: $\%_T$ compares factorization running times ($<$ 100 means that WS+COAMD is better), $\%_F$ compares flops, and $\%_Z$ compares number of nonzeros in the factors. The last column, denoted $\%_O$, show the time to find wide-separators as a percentage of the WS+COLAMD factorization time.

Name	MSMD	WS+ MSMD	COLMMD	COLAMD	WS+ COLAMD	$\%_T$	$\%_F$	$\%_Z$	$\%_O$
bwm2000	**2.75E+04**	7.86E+04	**2.75E+04**	2.86E+04	2.83E+04	200	98	98	100
cavity04	9.57E+05	9.57E+05	1.30E+06	**6.37E+05**	**6.37E+05**	100	100	100	0
poli_large	**1.45E+05**	**1.45E+05**	1.65E+05	1.70E+05	1.70E+05	100	100		37
bayer10	1.04E+07	3.01E+07	1.24E+07	**1.01E+07**	1.45E+07	125	143		2040
lhr04c	**1.47E+07**	6.73E+07	1.68E+07	1.77E+07	3.25E+07	164	183	128	150
bayer02	1.09E+07	1.09E+07	9.72E+06	**9.28E+06**	**9.28E+06**	117	100		362
rw5151	3.12E+07	3.16E+07	3.29E+07	3.29E+07	**2.92E+07**	92	88	92	37
lhr07c	**2.78E+07**	2.22E+08	3.16E+07	3.06E+07	6.67E+07	196	217	135	132
bayer04	2.79E+07	2.79E+07	**2.41E+07**	2.51E+07	2.51E+07	100	100		447
lhr10c	3.72E+07	**3.32E+08**	3.98E+07	3.92E+07	1.31E+08	197	334	152	533
lhr11c	**4.77E+07**	**4.77E+07**	5.18E+07	5.22E+07	5.22E+07	116	100	100	343
memplus	**3.95E+07**	**3.95E+07**	4.01E+07	5.60E+09	5.60E+09	94	100	100	0
ex19	9.45E+07	1.12E+08	7.08E+07	**4.07E+07**	1.09E+08	230	267	151	83
lhr14c	8.68E+07	2.10E+09	**8.46E+07**	8.51E+07	2.60E+08	191	305	149	284
bayer01	6.12E+07	4.82E+08	6.47E+07	**4.76E+07**	1.11E+08	121	233		8857
ex35	1.03E+08	1.33E+08	9.25E+07	**5.65E+07**	1.38E+08	207	244	136	34
cavity26	1.77E+08	**1.39E+08**	1.71E+08	2.04E+08	1.48E+08	75	72	85	19
epb1	1.47E+08	1.22E+08	**1.02E+08**	1.43E+08	1.25E+08	116	87	95	27
goodwin	6.42E+08	5.77E+08	**5.06E+08**	1.91E+09	6.44E+08	34	33	57	15
epb2	7.14E+08	5.17E+08	7.14E+08	6.46E+08	**5.64E+08**	107	87	97	15
garon2	1.18E+09	1.20E+09	1.28E+09	**1.06E+09**	1.98E+09	184	186	119	5
shyy161	1.07E+09	9.00E+08	1.04E+09	1.03E+09	**7.56E+08**	77	73	92	34
graham1	1.69E+09	**9.24E+08**	1.42E+09	1.33E+09	9.54E+08	72	71	82	11
epb3	2.22E+09	**8.09E+08**	1.79E+09	2.06E+09	1.18E+09	77	57	83	27
olafu	3.16E+09	2.71E+09	2.96E+09	2.84E+09	**2.58E+09**	73	90	89	21
rim	2.89E+09	2.01E+09	2.12E+09	5.55E+09	**1.77E+09**	31	31	54	28
venkat50	**4.30E+09**	4.36E+09	5.84E+09	4.51E+09	4.91E+09	93	108	85	13
venkat25	**4.30E+09**	4.36E+09	5.84E+09	4.51E+09	4.91E+09	94	108	85	14
venkat01	**4.30E+09**	4.36E+09	5.79E+09	4.46E+09	4.87E+09	93	109	85	14
ex40	3.69E+09	3.39E+09	2.29E+09	**1.08E+09**	2.87E+09	268	265	146	8
af23560	5.33E+09	7.01E+09	4.95E+09	**4.52E+09**	9.50E+09	181	210	133	1
raefsky3	1.05E+10	**5.24E+09**	7.75E+09	1.04E+10	5.47E+09	45	52	64	11
ex11	1.55E+10	1.19E+10	1.43E+10	1.19E+10	**1.12E+10**	76	94	92	6
raefsky4	1.56E+10	**7.80E+09**	1.07E+10	1.10E+10	8.56E+09	62	77	80	9
psmigr_1	**1.48E+10**	**1.48E+10**	1.66E+10	1.68E+10	1.68E+10	94	100	100	0
psmigr_3	**1.58E+10**	**1.58E+10**	1.72E+10	1.74E+10	1.74E+10	95	100	100	0
psmigr_2	**1.56E+10**	**1.56E+10**	1.74E+10	1.76E+10	1.76E+10	94	100	100	0
wang3	3.12E+10	**1.55E+10**	3.47E+10	2.78E+10	2.45E+10	84	88	90	0
wang4	3.70E+10	**2.45E+10**	3.52E+10	3.37E+10	2.72E+10	81	80	89	0
bbmat	5.97E+10	4.77E+10	**4.46E+10**	**4.46E+10**	5.82E+10	109	130	112	4
li	1.59E+11	**8.10E+10**	2.17E+11	1.63E+11	8.15E+10	44	50	72	4

degree orderings, but the cost is typically small relative to the cost of the subsequent factorization.

The use of the block triangular decomposition of the matrices and ordering seems to resolve the problems with some of the chemical engineering problems, but we are still investigating this issue.

Acknowledgments: Thanks to John Gilbert for telling us about wide-separator orderings. Thanks to John Gilbert and Bruce Hendrickson for helpful comments on an ealy draft of the paper. Thanks to Cleve Ashcraft for his encouragement, for numerous discussions concerning this research, and for his prompt response to our questions concerning SPOOLES.

References

1. Cleve Ashcraft and Roger Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, San-Antonio, Texas, 1999. 10 pages on CD-ROM.
2. T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse lu factorization. *SIAM Journal on Matrix Analysis and Applications*, 19:140–158, 1997.
3. T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, 25:1–19, 1999.
4. James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20:720–755, 1999.
5. A. George and J. W. H. Liu. The evolution of the minimum-degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.
6. Alan George and Esmond Ng. On the complexity of sparse QR and LU factorization on finite-element matrices. *SIAM Journal on Scientific and Statistical Computation*, 9:849–861, 1988.
7. John R. Gilbert. *Graph Separator Theorems and Sparse Gaussian Elimination.* PhD thesis, Stanford University, 1980.
8. John R. Gilbert. Predicting structure in sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 15:62–79, 1994.
9. John R. Gilbert and Robert Schreiber. Nested dissection with partial pivoting. In *Sparse Matrix Symposium 1982: Program and Abstracts*, page 61, Fairfield Glade, Tennessee, October 1982.
10. S. I. Larimore. An approximate minimum degree column ordering algorithm. Master's thesis, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, Florida, 1998. Also available as CISE Tech Report TR-98-016 at ftp://ftp.cise.ufl.edu/cis/tech-reports/tr98/tr98-016.ps.
11. Xiaoye S. Li. *Sparse Gaussian Elimination on High Performance Computers.* PhD thesis, Department of Computer Science, UC Berkeley, 1996.
12. Alex Pothen and Chin-Ju Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, December 1990.