Nested-Dissection Orderings for Sparse LU with Partial Pivoting[*]

***Igor Brainman*[†] *and Sivan Toledo*[‡]**

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 [3, 4], 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 [7] 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 [9] later showed that this upper bound

[*]This research was supported by Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (grant number 572/00 and grant number 9060/99) and by the University Research Fund of Tel-Aviv University.

[†]School of Computer Science, Tel-Aviv University.

[‡]School of Computer Science, Tel-Aviv University. Email: stoledo@tau.ac.il

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 [6]). 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 [8, 10]. 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.

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-

[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.

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^TA, but without computing it. We find separators in A^TA 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^TA. 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^TA without computing G_{A^TA} (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 [11], but they are low enough to easily justify using the new method. On many matrices, including large ones, our method significantly 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^TA} 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 developing 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 3. *Let S be a separator in an undirected graph G. The sets $S_1 = S \cup \{i | i \in V_1, (i, j) \in E \text{ for some } j \in S\}$ and $S_2 = S \cup \{i | i \in V_2, (i, j) \in E \text{ for some } j \in S\}$ 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

This section summarizes our experimental results. We begin by describing our code, our collection of test matrices, and the machine used to carry out the experiments. We then describe and analyze the results of our experiments. The analyses focus on the effectiveness of various ordering methods and on their performance. By effectiveness we mean the number of nonzeros in the factors, the number of floating-point operations (flops) required to compute them, and the factorization time. By performance we mean the cost, mostly in terms of time, of the ordering algorithm itself.

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 developed by Cleve Ashcraft and others. Our codes then invoke a column-minimum-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 [11] 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). In an earlier experiment, reported in [2], we also tested COLMMD, a column minimum-degree code, originally written by Joseph W.-H. Liu and distributed with SuperLU. It was not shown to be consistently superior to the other two codes so we dropped it from the experiment reported here.

We use the following acronyms 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 the experiments reported here, we always reduce the input matrices to *block triangular form* (see [13]) and factor only the diagonal blocks in the reduced form. Many of the matrices in our test suite have numerous tiny diagonal blocks (most of them 1-by-1); we report the performance of factoring all the diagonal blocks of size 250 or larger.

We factor the reordered matrix using SuperLU [5, 12] version 2.0, a state-of-the-art sparse-LU-with-partial-pivoting code. SuperLU uses the BLAS. we used ATLAS[2], a high-performance implementation of the BLAS.

We conducted the experiments on a 600MHz dual Pentium III computer with 2 GBytes of main memory running Linux. The machine was configured without swap space so no paging occurred during the experiments. 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[3]. We used all the nonsymmetric matrices in Davis's collection that are not too small (we only report results on matrices whose factorization time with the best ordering method was at least 1 second). Two of the matrices in Davis's collection were too large to factor on our machine (appu and pre2) and SPOOLES broke down on a third (av41092; we are unsure whether the breakdown is due to a bug in our code or due to a problem in SPOOLES). The matrices are listed in Table 1. For further details about the matrices, see Davis's

[2]`www.netlib.org/atlas`

[3]`www.cise.ufl.edu/~davis/sparse/`

Table 1. *A comparison of wide-separator and minimum-degree orderings on matrices whose graphs are regular 2- and 3-dimensional meshes. The numerical values in the matrices are uniform random variables in* $[0, 1]$*. The first two lines show the results on 2D meshes, the rest on 3D meshes. A missing timing result means that the factorization ran out of space (all were executed on the same 2GB machine). Times are in seconds.*

N_x	N_y	N_z	T^F_{best}	Best Method	$T^{F+O}_{\text{wscolamd}}$	T^{F+O}_{colamd}	$T^{F+O}_{\text{wscolmsmd}}$	T^{F+O}_{colmsmd}
500	500		113	wscolamd	150	202	150	—
750	750		496	wscolamd	601	—	684	—
30	30	30	352	colamd	399	352	1210	404
40	40	40	786	wscolamd	792	2340	958	—

web site.

We also run experiments on matrices whose graphs are regular 2- and 3-dimensional meshes and whose values are random numbers in the range $[0, 1]$.

3.2 Results and Analysis

Tables 1 and 2 summarize the results of our experiments.

Table 2 shows that wide-separator (WS) orderings are both effective. On the largest 2D and 3D meshes WS orderings lead to the fastest factorization times and to the fastest overall solution time (including ordering time). Beyond performance, WS orderings enable us to solve problem that we could simply not solve with minimum-degree orderings since the factorization runs out of space.

Table 1 summarizes the results of our experiments on matrices from a matrix collection. Column 5–10 in the table show that wide-separator ordering techniques are effective. Wide separator (WS) orderings are the most effective ordering methods, in terms of the factorization time, on 16 out of the 32 test matrices. WS orderings are the most effective on 7 out of the 10 largest matrices (largest in terms of factorization time).

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

Nonzero counts in the factors, which are not shown in the table, are generally correlated with factorization time, so a shorter factorization time for a matrix also imply smaller factors.

When WS orderings do poorly compared to MD methods, however, they sometimes do significantly poorer. On `ex40`, for example, using wide separators slows down the factorization by a factor of 2.45 relative to COLAMD alone. The In earlier experiments [2] in which we did not reduce the matrices to block triangular form, we have found that on some of matrices, especially the `lhr` and `bayer` matrices, the slowdowns are even more dramatic. The experiments reported here show that reduction to block triangular form resolves these problems.

Wide-separator orderings are more expensive to compute than strict minimum-degree orderings. Table 1 shows that when the ordering times are taken into account, wide-separator orderings speed up the total solution time in 9 out of the 32 matrices. In 7 other matrices a wide-separator ordering reduced the factorization time but increased the total solution time. We note that even when a wide-separator ordering only reduces the factorization time, it also reduces the size of the factors, which is often highly important (since it saves memory, reduces the occurrence of paging, and speeds up subsequent triangular solves).

4 Conclusions And Discussion

Our main conclusion from this research is that hybrid wide-separator/minimum-degree column orderings are effective. WS orderings are clearly superior to minimum-degree orderings alone on large 2D and 3D meshes that require pivoting. On other matrices obtained from a matrix collection, WS orderings often reduce substantially the amount of time and storage required to factor a sparse matrix with partial pivoting, compared to column-MD orderings. They are more expensive to compute than minimum-degree orderings but the expense is often more than payed off by reductions in time and storage during the factorization stage.

Wide-separator orderings, like other column orderings based on fill in the factors of A^TA, are robust but pessimistic. They are robust in the sense that they reduce worst-case fill. Optimistic column orderings that attempt to reduce the fill in the factors of $A^T + A$ tend to reduce fill better than pessimistic orderings when little or no pivoting occurs, but can cause catastrophic fill when pivoting does occur.

The combined results of this paper and of an earlier paper [2] show that first permuting the matrix to block triangular form reduces the wide-separator ordering times and improves the quality of the ordering on some matrices.

This work can be extended in several directions. First, improving the performance of the ordering phase itself would be significant. This can be done by tuning the parameters of the ordering code (stopping the recursive bisection on fairly large subgraphs) or by improving the wide-separator algorithm itself. Second, one can try to improve the orderings by trying to derive smaller wide-separators from a given conventional separator. Third, one can interleave the ordering and factorization in a way that widens separators only when necessary. That is, we would find a conventional separator S in G, recursively order G_1 and factor the columns corresponding to G_1. Once this phase is completed, we can widen the separator by adding to S the neighbors of vertices that were used as pivots. We now recursively order and factor the (shrunken) G_2.

Acknowledgments: Thanks to John Gilbert for telling us about wide-separator orderings. Thanks to John Gilbert and Bruce Hendrickson for helpful comments on an early 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.

Table 2. *A comparison of wide-separator and minimum-degree column orderings. Column 2 shows the number of blocks in the block-triangular form of the matrix and column 3 the number of blocks larger than* 250*-by-*250*. Column 4 shows the fastest factorization time and column 5 shows the ordering method that led to the fastest time. Column 6 shows the ratio of the factorization time using WSCOLAMD ordering relative to the factorization time using COLAMD. Values smaller than* 1 *mean that the wide-separator ordering reduced the factorization time. Column 7 shows the ratio of the sums of the factorization and ordering times. A value smaller than* 1 *in column 6 but larger than* 1 *in column 7 mean that the wide-separator ordering reduced the factorization time, but due to the overhead of the factorization itself, the total solution time increased. Columns 8 and 9 are similar to 6 and 7 with MSMD replacing COLAMD.*

					COLAMD		COLMSMD	
Name	N_{BTF}	$N_{\text{BTF}}^{>250}$	T_F^{best}	Best Method	$\frac{T_F^{\text{WS}}}{T_F}$	$\frac{T_{F+O}^{\text{WS}}}{T_{F+O}}$	$\frac{T_F^{\text{WS}}}{T_F}$	$\frac{T_{F+O}^{\text{WS}}}{T_{F+O}}$
bayer01	8861	1	1.01e+00	colamd	1.30	3.25	1.73	1.84
ex35	173	4	1.03e+00	colamd	1.57	4.09	1.14	3.62
utm5940	147	1	1.15e+00	colamd	1.17	1.48	1.40	1.34
epb1	1	1	1.41e+00	colamd	1.00	1.40	1.05	1.30
lhr34	3533	10	2.06e+00	colamd	1.39	5.83	1.52	7.87
lhr34c	3533	10	2.08e+00	colamd	1.49	6.00	1.62	7.98
shyy161	25761	1	4.55e+00	wscolamd	0.62	1.51	0.74	2.05
goodwin	2	1	4.73e+00	wscolamd	0.36	0.70	1.21	1.41
epb2	1	1	5.43e+00	wscolamd	0.93	1.18	0.89	1.10
raefsky2	1	1	6.80e+00	wscolmsmd	0.39	0.55	0.59	0.87
raefsky1	1	1	6.81e+00	wscolmsmd	0.39	0.55	0.59	0.87
graham1	478	1	8.80e+00	wscolamd	0.96	1.43	1.00	1.12
ex40	1	1	9.18e+00	colamd	2.45	2.50	0.88	0.78
garon2	1	1	9.44e+00	colamd	1.82	1.93	1.00	0.99
epb3	1	1	1.02e+01	wscolmsmd	0.69	1.03	0.49	0.97
rim	2	1	1.69e+01	wscolamd	0.36	0.72	1.14	1.34
olafu	1	1	2.02e+01	wscolmsmd	0.86	1.04	0.86	1.03
rma10	1	1	3.25e+01	colmsmd	0.96	1.25	1.15	1.31
venkat01	1	1	3.44e+01	colmsmd	1.00	1.18	1.01	1.21
venkat25	1	1	3.46e+01	colmsmd	0.99	1.18	1.01	1.20
venkat50	1	1	3.46e+01	colmsmd	0.99	1.17	1.01	1.20
raefsky3	1	1	3.85e+01	wscolmsmd	0.52	0.61	0.51	0.65
af23560	1	1	4.00e+01	colamd	1.98	1.90	1.31	1.25
raefsky4	1	1	5.83e+01	wscolmsmd	0.78	0.86	0.49	0.59
ex11	1	1	8.44e+01	wscolamd	0.93	1.00	0.76	0.83
psmigr-2	1	1	1.43e+02	wscolamd	0.99	1.03	0.99	1.00
psmigr-3	1	1	1.43e+02	wscolamd	0.99	1.03	1.02	1.01
psmigr-1	1	1	1.52e+02	colamd	1.01	1.05	0.97	1.01
wang3	1	1	1.65e+02	wscolmsmd	0.95	0.95	0.59	0.60
wang4	1	1	2.26e+02	wscolmsmd	0.89	0.90	0.71	0.72
bbmat	1	1	4.08e+02	colamd	1.24	1.31	0.76	0.72
li	2	2	6.96e+02	wscolamd	0.51	1.05	0.35	0.73

Bibliography

[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] Igor Brainman and Sivan Toledo. Nested-dissection orderings for sparse LU with partial pivoting. *Proceedings of the 2nd Conference on Numerical Analysis and Applications*, 8 pages, Rousse, Bulgaria, June 2000. (To appear in a forthcoming Springer LNSC volume.)

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] John R. Gilbert. *Graph Separator Theorems and Sparse Gaussian Elimination.* PhD thesis, Stanford University, 1980.

[9] John R. Gilbert. Predicting structure in sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 15:62–79, 1994.

[10] 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.

[11] 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.

[12] Xiaoye S. Li. *Sparse Gaussian Elimination on High Performance Computers.* PhD thesis, Department of Computer Science, UC Berkeley, 1996.

[13] 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.