Effective Sparse Matrix Ordering: Just Around the BEND*

Bruce Hendrickson† Ed Rothberg‡

Abstract

When performing sparse matrix factorization, the ordering of matrix rows and columns has a dramatic impact on the factorization time. This paper describes an approach to the reordering problem that produces significantly better orderings than previous methods. The algorithm is a hybrid of nested dissection and minimum degree ordering, and combines an assortment of algorithmic techniques.

1 Introduction

When using a direct method to solve a linear system $Ax = b$, where A is sparse, symmetric and positive definite, the first step is typically a heuristic reordering of the rows and columns of A to reduce *fill* in the factor matrix L (where the fill is the set of zero entries in A that become nonzero in L). Limiting fill generally reduces the amount of work and storage required to factor A. The most commonly used heuristic for performing reordering is the minimum degree algorithm [13, 30]. An alternative approach, nested dissection ordering [11, 12], has many appealing theoretical properties, but building an implementation that gives comparable ordering qualities and runtimes to the minimum degree method has proven to be quite difficult. Some promising results have been demonstrated [3, 8, 18, 21, 27], but unfortunately none of these previous efforts have produced consistently better orderings than minimum degree, and all require significantly more runtime.

An important difference between minimum degree and nested dissection is that minimum degree can be described concisely: it always eliminates a column containing the fewest non-zero values next. Variations in implementation are possible (e.g., Multiple Minimum Degree (MMD) [22] and Approximate Minimum Degree (AMD) [1]), but the fundamental algorithm remains unchanged. The nested dissection "algorithm", on the other hand, is in fact quite ill-specified. The method is instead a general ordering framework. Nested dissection is fundamentally a divide and conquer approach, where division is accomplished by finding vertex separators in a graph associated with sparse matrix A. Questions left unaddressed include:

- What properties should the vertex separators have?
- What algorithms should be used to find them?

*This paper appears in the Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing, and is a condensation of two technical reports [17, 29].

†Sandia National Labs; bahendr@cs.sandia.gov. Supported by the Mathematical, Information and Computational Sciences Division of the U.S. DOE, Office of Energy Research, and works at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AC04-94AL85000. Sandia is multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

‡Silicon Graphics, Inc.; rothberg@sgi.com.

- When should the nested recursion be halted?
- How should the separators be ordered?

In exploring these issues, we have developed a hybrid ordering strategy and a code we call BEND[1] which produces significantly better orderings than minimum degree and previous nested dissection approaches. The key advance behind our approach is a new method for finding separators, although we have also improved on several other recently proposed ideas. In the end, we obtain orderings that are significantly better than those generated by previous approaches.

We should note that ours is not the only recent work that has improved on previous nested dissection methods. Independent work by Ashcraft and Liu [5] and by Gupta [15] has also produced significantly better results.

2 Methods

The concepts and techniques used in our ordering approach are described in the following subsections. In §2.8 we explain how these pieces fit together to comprise our algorithm.

2.1 Preliminaries

Algorithms involving sparse matrices are often most naturally described as operations on graphs. A *graph* $G = (V, E)$ consists of a set of *vertices* V and a set of vertex pairs E known as *edges*. We denote vertices by i, and we use e_{ij} for an edge between vertices i and j. We use $Adj(i)$ to refer to the set of vertices adjacent to vertex i. The *degree* of a vertex is the number of edges incident to it. Each vertex and each edge can also have an associated value known as a *weight*, which we denote by $w(i)$ and $w(e_{ij})$ respectively. The nonzero structure of a symmetric, sparse matrix A can be conveniently represented by a graph. The n vertices of the graph correspond to the n columns of the matrix, and an edge connects vertices i and j if $A(i, j)$ is nonzero.

The evolution of the nonzero structure during Cholesky factorization can be described succinctly in terms of the graph G of A. In graph terms, the elimination of vertex i creates edges e_{jk} in G for every pair of vertices j and k in $Adj(i)$.

The added edges in this process correspond precisely to the *fill* in the matrix. The number of floating-point operations required to perform the factorization is equal to the sum of the squares of the degrees of each eliminated vertex. Clearly, these quantities are functions of the order in which the rows and columns are eliminated. The goal of reordering techniques is to reduce the fill and the floating-point complexity of the factorization, which leads to faster, less memory-intensive solution.

2.2 Minimum Degree and Nested Dissection Ordering

Since the number of operations incurred when eliminating a column is the square of the number of nonzeros in the column, a reasonable heuristic strategy would be to always eliminate the vertex with the smallest degree. This simple greedy strategy is the basis of a very successful reordering heuristic known as the *minimum degree* ordering [25, 30]. Due to the quality of orderings produced by minimum degree and the fast run times of the advanced variants [1, 22], minimum degree methods are by far the most popular methods for reordering sparse matrices. However, they are poorly understood theoretically, and their worst case behavior can be far from optimal [7].

[1] Bruce and Ed's Nested Dissection

An alternative to minimum degree was proposed by George [11], and is known as *nested dissection*. The basic idea is to find a set of columns S, whose removal decouples the matrix into two parts, X and Y, whose nonzero values are in disjoint rows and columns. If we order S after X and Y, then no fill can occur in the off-diagonal blocks of the submatrix consisting of X and Y. Once a set S has been found, X and Y can be reordered by applying the dissection strategy recursively, or by any other technique (e.g. minimum degree). However, if minimum degree is used on the subgraphs, the quality of the ordering is enhanced by accounting for the existence of the neighboring separator vertices. In particular, it is useful to recognize that fill can occur between the vertices in X (or Y) and those in S. *Constrained* minimum degree is an enhancement to pure minimum degree which can take this into consideration [24].

As with minimum degree, nested dissection can be described in terms of graph operations on the graph of the matrix. The set S is a vertex separator of the graph, and X and Y are the two pieces separated by S. One key to an effective nested dissection strategy is thus the ability to find small vertex separators.

As our results will demonstrate, a second key is allowing some imbalance between the sizes of the two separated pieces X and Y. The utility of this tolerance was first observed by Lewis and Leiserson [21], but has been largely overlooked. To make progress in a recursive divide-and-conquer approach, neither X nor Y can be too small, but precise equality seems to be an unnecessary constraint. A substantially better separator can often be found if some imbalance is tolerated.

Despite some attractive theoretical results (eg. [11]), nested dissection until recently has not been competitive in practice. Orderings produced by nested dissection have been less consistent and usually poorer than those produced by minimum degree, and nested dissection algorithms are generally much slower than MMD.

2.3 Compressed Graphs

Matrices that arise in many application areas, most notably finite element analysis, can contain multiple columns with identical adjacency structures. In graph terms, the graph G of A contains sets of vertices where, for any two members of the set i and j, $Adj(i) \cup \{i\} = Adj(j) \cup \{j\}$. One can form a *compressed graph* by merging all vertices with the same adjacencies into a single, weighted vertex, where the weight is equal to the number of original vertices merged into that compressed vertex. The minimum degree and nested dissection methods can easily be modified to work on the smaller, compressed representation of the graph [2]. The cost of computing the compressed graph is $O(|E|)$ (see [2, 17] for details), and is quite small in practice.

2.4 Multilevel Partitioning Algorithms

In recent years, multilevel techniques for finding edge separators in graphs have received a great deal of attention. First proposed by Bui and Jones [8], and improved by Hendrickson and Leland [16] and Karypis and Kumar [18], these methods have proved capable of finding high quality edge separators very quickly. The basic idea has a close analogy with multigrid methods from continuous mathematics. The original graph is approximated by a sequence of smaller and smaller graphs. The smallest graph in the sequence is partitioned into two disjoint sets, X and Y, so as to minimize the number of edges between the sets. This partition is propagated back through the sequence of graphs, while being refined periodically.

This general partitioning framework has been used in several nested dissection implementations [8, 19], including Karypis and Kumar's METIS software [19] which we compare against in §3. These methods first find an edge separator via a multilevel algorithm and then derive a vertex separator from the edge separator, typically using a matching technique that we review in §2.6. This approach has the limitation that the quantity it attempts to minimize, the size of the edge separator, is only indirectly related to the quantity that should be minimized, the size of the vertex separator.

To avoid this problem, we use a multilevel strategy that finds vertex separators directly. To do this, we require a partitioning algorithm that finds and/or refines a vertex separator instead of an edge separator. We use two refinement strategies for this problem: vertex Fiduccia-Mattheyses which we discuss in §2.5, and a maximum-flow technique which we describe in §2.6. This latter algorithm is a generalization to weighted graphs of the matching technique that is used to derive vertex separators from edge separators. A closely related multilevel algorithm has been independently developed by Gupta [15].

2.5 Kernighan-Lin/Fiduccia-Mattheyses for Refining Vertex Separators

An important class of algorithms for refining edge separators was introduced by Kernighan and Lin [20] and improved by Fiduccia and Mattheyses [10]. The fundamental idea behind these algorithms is the notion of *gain*, which is the reduction in the size of the separator associated with moving a vertex from one set to another. Kernighan and Lin coupled this simple concept with an innovative local search strategy to produce a widely imitated algorithm. Their insight was to repeatedly move the vertex with the greatest gain, even if that gain was negative, meaning that the move would make the partition worse. Their hope was that allowing some unhelpful moves might allow for the discovery of better partitions a few moves hence. The best partition encountered this way is recorded. Fiduccia and Mattheyses modified the original algorithm so that a single search for a better partition runs in time bounded by the number of edges in the graph. This algorithm has generally been used to find edge separators, so we will denote it as *edge FM* or *EFM*.

The algorithm consists of two nested loops. The inner loop performs a sequence of trial moves, remembering the best partition ever encountered. Since the algorithm allows for moves that make the partition worse, the possibility of infinite looping exists, so the constraint is imposed that a vertex cannot move twice within the inner loop. This constraint motivates the need for the outer loop that begins the process over again. In practice, the outer loop is executed a small number of times (≤ 10), particularly if the algorithm is initialized with a good partition (as typically happens when used in a multilevel method).

With appropriate data structures, an inner loop can be executed in time proportional to the number of edges in the graph. The initial gain values can be computed in $O(|E|)$ time, and sorted with a bucket sort. The best move can now be determined in constant time. In EFM when a vertex is moved, only the gains of its neighbors are affected and their new values can be computed and moved to new buckets in constant time. In this way the overall time for an inner loop is bounded by the sum of the vertex degrees, or $O(|E|)$.

Most prior partitioning work has been devoted to improving edge separators, but for our application we need to refine a vertex separator. Fortunately, the FM approach was extended by Ashcraft and Liu [3] to address this problem in a straightforward manner. The method begins with three sets of vertices, a separator S that divides the remaining vertices into two disconnected pieces X and Y. The moves we consider take a vertex v from S and transfer it to X (or Y). When we do this, the neighbors of v that were in Y (or X) are pulled into S. With this basic operation, the structure of the FM algorithm with inner and outer loops can be preserved. We will call this algorithm *vertex FM* or *VFM*.

2.6 Minimum Weight Vertex Cover of a Bipartite Graph

Liu [23] described an alternate algorithm for refining a vertex separator. In contrast to the VFM approach, which moves one vertex out of the separator at a time, this refinement algorithm potentially moves a number of vertices simultaneously. If we define X_S to be the set of vertices in X adjacent to vertices in S, then the algorithm finds a minimum number of vertices that forms a vertex cover for the edges between S and X_S. This same approach can be used to obtain a vertex separator from an edge separator. One simply computes a minimum vertex cover for the separator edges [26].

This refinement algorithm has generally been applied to unweighted graphs, where it must find a minimum cardinality vertex cover. Since we wish to apply this refinement algorithm at multiple levels of the multilevel method, we must instead find a minimum *weighted* vertex cover. While the unweighted case is most often described as a maximum bipartite matching problem [26], both the unweighted and weighted vertex cover problems can also be described as network flow problems [6]. More details can be found in [4, 17].

In our experiments, we apply this minimum weighted vertex cover approach (which we refer to as WVC) in an iterated fashion. We continue applying the weighted cover technique until we can find no smaller separator.

2.7 Combining Nested Dissection and Minimum Degree

In order to improve both runtimes and ordering qualities, we actually use a hybrid of minimum degree and nested dissection. We hybridize the methods in two ways. The first is the standard incomplete nested dissection method [14]. Starting with the original graph, we perform several levels of nested dissection. Once the subgraphs are smaller than a certain size, we order them using minimum degree. This allows us to reap the benefits of nested dissection at the top levels, where most of the factorization work is performed, while obtaining the runtime advantages of minimum degree on the smaller problems.

The second hybridization we use can be thought of as minimum degree post-processing on an incomplete nested dissection ordering. This method was originally proposed independently by Ashcraft and Liu [5, 6] and by Rothberg [28]. The idea is to reorder the separator vertices using minimum degree. A simple intuition behind this hybrid method is that nested dissection makes an implicit assumption that recursive division of the problem is the best approach to ordering. Allowing minimum degree to reorder the separator vertices removes this assumption.

2.8 Bringing it All Together

Our reordering algorithm begins by attempting to compress the graph as described in §2.3. We then use an incomplete nested dissection algorithm to reorder the (possibly weighted) graph. We apply nested dissection as described in §2.2 until the pieces are fairly small (at most $n/32$ vertices). Since minimum degree is generally quite effective for modest-sized graphs, we invoke constrained Approximate Minimum Degree (AMD) as discussed in §2.2 on the subgraphs.

Our algorithm uses a multilevel algorithm to find vertex separators for nested dissection. As discussed in §2.4, this approach finds a vertex separator directly. We coarsen until there are fewer than 50 vertices in the graph, and then find an initial separator by placing every vertex in the separator and applying VFM as described in §2.5. As mentioned in §2.2, we allow significant imbalance between subgraphs in the course of computing separators.

As the separator is propagated through the sequence of intermediate graphs, we refine it every two uncoarsening stages using VFM followed by iterated WVC as described in §2.6. Finally, we apply a post-pass of AMD to all the separator vertices.

3 Results

Due to space limitations, we can only briefly summarize our results here. The interested reader is directed to [17, 29] for many more details.

We applied BEND to a variety of matrices from structures analysis, fluid dynamics and linear programming. The structures and fluids problems included the largest problems from the Harwell-Boeing collection [9], the NAS graph collection at NASA Ames, a structural analysis application by Storaasli at NASA Langley and SGI customer applications. Three of these 25 matrices come from 3D applications and the remainder from 2D or 2.5D. The 22 linear programming problems come from NETLIB and from SGI customer applications. The test suite was selected before any of the experiments were performed.

We compared our results to three widely used reordering codes: Liu's implementation of MMD [22], the AMD implementation of Amestoy, Davis and Duff, and METIS, a state-of-the-art nested dissection code by Karypis and Kumar [19]. We use geometric means to aggregate the results over the suite of test problems.

Our primary metric of ordering quality was the number of floating point operations required to perform the factorization, but we also monitored the number of nonzero values in the lower triangle of the factored matrix.

Observation 1. In aggregate, the orderings produced by AMD are of the same quality as those due to MMD, but AMD is significantly faster. Specifically, AMD averaged 2.3 times faster than MMD on the 2D problems, 1.2 times faster on the 3D problems and 13.7 times faster on the LP problems.

Since AMD looks to be the minimum degree variant of choice, we report our additional results normalized against AMD. For each problem, we divided the run-time/operations/storage against those for AMD, and then took geometric means across problems.

Observation 2. Compression significantly reduces the runtime of our ordering approach without harming (and occasionally helping) quality. On the suite of structures and fluids problems, it reduced runtime by an average of about a factor of two. However, the LP problems were not amenable to compression. Our particular (small) set of 3D problems came from scalar equations and so did not compress either.

Observation 3. It is important to compute vertex separators directly, rather than deriving a vertex separator from an edge separator. The vertex separator approach produced better orderings in about the same time. The improvement was 6% for the 2D problems, 19% for the 3D problems and 48% for the LP problems.

Observation 4. It is crucial to allow some imbalance in the separators. In other words, the subgraphs that remain when the separator is removed should not be required to be of equal size. Allowing imbalance improved orderings by 13%, 21% and 22% respectively for the 2D, 3D and LP problems with a minimal increase in runtime.

Observation 5. Combining two local improvement strategies (vertex FM and iterated weighted vertex cover) proved to be more effective than using just one (vertex FM). Orderings improved by another 13%, 13% and 8% for the 2D, 3D and LP problems.

However, ordering times went up by 10–20%.

Observation 6. Using minimum degree to reorder the separator vertices was a significant improvement over nested dissection alone, with minimal impact on runtime. For the 2D, 3D and LP problems, this improved orderings by a further 12%, 8% and 9%.

Observation 7. For parallel factorization, an additional relevant metric is the height of the factor elimination tree (e-tree), which corresponds to the critical path in the factorization. Shorter e-trees are preferable for parallel factorization. Our overall method reduced e-tree heights by 20%, 47%, and 25% relative to AMD for the 2D, 3D, and LP problems. Not surprisingly, reordering the separator vertices using AMD had a negative impact on e-tree heights. Forgoing this step produced significantly greater reductions in e-tree heights (48%, 64%, and 25%). Somewhat to our surprise, the introduction of imbalance in the separators had a negligible impact on on-tree heights.

Conclusion. Overall, our approach generated orderings that were significantly better than those produced by minimum degree or METIS for all problem classes as evidenced by Table 1 below.

TABLE 1

Geometric averages of results relative to AMD.

	2D Grids	3D Grids	LP Matrices
Operations			
MMD	.99	.95	1.00
METIS	1.04	.34	1.13
BEND	.69	.24	.50
Storage			
MMD	.99	.98	1.00
METIS	1.08	.62	1.13
BEND	.87	.56	.74
Runtime			
MMD	2.4	1.3	13.7
METIS	14.1	5.4	1.3
BEND	6.0	5.4	3.8

Acknowledgements

We are indebted to Cleve Ashcraft for several helpful conversations.

References

[1] P. Amestoy, T. Davis, and I. Duff, *An approximate minimum degree ordering algorithm*, Tech. Rep. TR-94-039, University of Florida, December 1994.

[2] C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM J. Sci. Comput., 16 (1995), pp. 1404–1411.

[3] C. Ashcraft and J. W. H. Liu, *A partition improvement algorithm for generalized nested dissection*, Tech. Rep. BCSTECH-94-020, Boeing Computer Services, 1994.

[4] C. Ashcraft and J. W. H. Liu, *Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement*, Tech. Rep. ISSTECH-96-017, Boeing Computer Services, 1996.

[5] C. Ashcraft and J. W. H. Liu, *Robust ordering of sparse matrices using multisection*, Tech. Rep. ISSTECH-96-002, Boeing Information and Support Services, 1996.

[6] ——, *Generalized nested dissection: some recent progress.* Minisymposium presentation at the Fifth SIAM Conference on Applied Linear Algebra, Snowbird, Utah, June 18, 1994.

[7] P. Berman and G. Schnitger, *On the performance of the minimum degree algorithm for Gaussian elimination*, SIAM J. Matrix Analysis and Applic., 11 (1990), pp. 83–88.

[8] T. Bui and C. Jones, *A heuristic for reducing fill in sparse matrix factorization*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 445–452.

[9] I. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Soft., 15 (1989), pp. 1–14.

[10] C. M. Fiduccia and R. M. Mattheyses, *A linear time heuristic for improving network partitions*, in Proc. 19th IEEE Design Automation Conference, IEEE, 1982, pp. 175–181.

[11] A. George, *Nested dissection of a regular finite element mesh*, Tech. Rep. STAN-CS-208, Stanford University, 1971.

[12] A. George and J. W. H. Liu, *Computer solution of large sparse positive definite systems*, Prentice Hall, 1981.

[13] ——, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[14] A. George, J. W. Poole, and R. Voigt, *Incomplete nested dissection for solving n by n grid problems*, SIAM J. Numer. Anal, 15 (1978), pp. 663–673.

[15] A. Gupta, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, Tech. Rep. RC 20496 (90799), IBM T. J. Watson Research Center, Yorktown Heights, NY, July 10, 1996. To appear in *IBM Journal of Research and Development*.

[16] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, in Proc. Supercomputing '95, ACM, November 1995.

[17] B. Hendrickson and E. Rothberg, *Improving the runtime and quality of nested dissection ordering*, Tech. Rep. 96–0868, Sandia National Labs, March 1996. Submitted for publication.

[18] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Rep. TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[19] ——, *METIS: Unstructured graph partitioning and sparse matrix ordering system*, tech. rep., Department of Computer Science, University of Minnesota, 1995.

[20] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal, 29 (1970), pp. 291–307.

[21] C. Leiserson and J. Lewis, *Ordering for parallel sparse symmetric factorization*, in Proc. SIAM Conf. on Parallel Processing for Scientific Computing, 1987, pp. 27–31.

[22] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Soft., 11 (1985), pp. 141–153.

[23] ——, *A graph partitioning algorithm by node separators*, ACM Trans. Math. Software, 15 (1989), pp. 198–219.

[24] ——, *The minimum degree ordering with constraints*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1136–1145.

[25] H. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.

[26] A. Pothen and C. Fan, *Computing the block triangular form of a sparse matrix*, ACM Trans. Math. Soft., 16 (1990), pp. 303–324.

[27] A. Pothen, H. Simon, L. Wang, and S. Barnard, *Toward a fast implementation of spectral nested dissection*, in Proc. Supercomputing '92, 1992, pp. 42–51.

[28] E. Rothberg, *Robust ordering of sparse matrices: a minimum degree, nested dissection hybrid.* Silicon Graphics manuscript, 1995.

[29] E. Rothberg and B. Hendrickson, *Sparse matrix ordering methods for interior point linear programming*, Tech. Rep. 96–0475, Sandia National Labs, January 1996. To Appear in INFORMS J. Comput.

[30] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, J. Proc. IEEE, 55 (1967), pp. 1801–1809.