Sparse Matrix Ordering Methods for Interior Point Linear Programming

Edward Rothberg
Silicon Graphics, Inc.
Mountain View, CA 94043

Bruce Hendrickson
Sandia National Laboratories
Albuquerque, NM 87185

January 31, 1996
Revised September 21, 1996

Abstract

The main cost of solving a linear programming problem using an interior point method is usually the cost of solving a series of sparse, symmetric linear systems of equations, $A\Theta A^T x = b$. These systems are typically solved using a sparse direct method. The first step in such a method is a reordering of the rows and columns of the matrix to reduce fill in the factor and/or reduce the required work. This paper evaluates several methods for performing fill-reducing ordering on a variety of large-scale linear programming problems. We find that a new method, based on the nested dissection heuristic, provides significantly better orderings than the most commonly used ordering method, minimum degree.

1 Introduction

An interior point method solves a linear programming problem by computing a sequence of direction vectors. At each iteration, the method takes a step in the computed direction, moving closer to the optimal solution. The details of the interior point method are not relevant to this paper, so we refer the reader to [21, 22] for more information. The dominant computation in each iteration of the method is the solution of a sparse linear system $A\Theta A^T x = b$ to determine the step direction. The matrix Θ, a diagonal matrix, changes from iteration to iteration, while the A matrix remains constant. The linear systems are typically solved by factoring the matrix $M = A\Theta A^T$ into $M = LDL^T$, where L is lower triangular and D is diagonal. The solution vector x is then computed by solving $Lz = b$ for z, then $Dy = z$ for y, then $L^T x = y$.

When matrix M is factored into LDL^T, the factor matrix L suffers some amount of *fill*: L_{ij} entries become non-zero where the corresponding M_{ij} are zero. Fill in the factor matrix can be quite substantial — a 50-fold increase in non-zeroes is not uncommon. The amount of fill is strongly influenced by the ordering of the rows and columns of M; factoring a permuted matrix PMP^T, where P is a permutation matrix, can dramatically reduce the amount of work required for the factorization. Finding the optimal ordering is an NP-complete problem, so the matrix is typically reordered using a heuristic. The most commonly used heuristic is *minimum degree ordering* [12, 24, 30]. Variants of minimum degree include Quotient Minimum Degree (QMD) [11], Multiple Minimum Degree (MMD) [20], and Approximate Minimum Degree (AMD) [1].

Minimum degree ordering is not the only available ordering heuristic. One important alternative is *nested dissection* [10, 11]. While early implementations of nested dissection (specifically Automated Nested Dissection) were much less effective than minimum degree, recent developments have changed this situation. Better methods for finding graph separators, which form the basis for nested dissection ordering, are now available. Examples include spectral methods [27, 28], multi-level Fiduccia-Mattheyses methods [7, 8, 14, 17, 18], and vertex-separator Fiduccia-Mattheyses variants [4, 15].

The most widely used ordering method in interior point linear programming today is multiple minimum degree ordering. This paper looks at five other ordering methods, investigating whether any of them offers significant advantages over MMD for a range of large-scale linear programming problems.

2 Sparse matrix ordering

This section describes the minimum degree and nested dissection ordering methods, including discussions of the variants we explore in this paper. All of the ordering methods we consider are most easily described in terms of the graph representation G of matrix M. Graph G has n vertices, where n is the number of rows and columns in M. There is an edge between vertices i and j in G for every non-zero M_{ij} in M. The *degree* of a vertex in G is equal the number of edges incident to that vertex. The sparse factorization is performed as a sequence of n *elimination* steps, where in each step i, vertex i is removed from the graph G and edges are added so that all neighbors of vertex i become adjacent in the new graph. The edges that are added represent fill in the factor matrix.

2.1 Minimum degree ordering

The first method we consider is minimum degree ordering. The intuition behind this method is quite simple. Since the elimination of a vertex causes its neighbors to become adjacent, minimum degree always chooses a vertex of minimum degree to eliminate next. Unfortunately, this very simple idea has historically proven to be quite difficult to implement efficiently [12]. Early implementations (e.g., Quotient Minimum Degree (QMD)) required enormous runtimes. Fortunately, several variants of minimum degree ordering have since been developed whose runtimes are quite reasonable in comparison to the cost of the subsequent factorization. We study three such variants. The first, Multiple Minimum Degree (MMD) [20], reduces the runtime of the algorithm by eliminating a set of vertices of minimum degree simultaneously. This multiple elimination technique dramatically reduces the cost of updating the degrees of the neighbors of eliminated vertices, the main cost of the algorithm. Whereas QMD must update neighbor degrees each time a vertex is eliminated, MMD will often eliminate many neighbors of a vertex before updating that vertex's degree.

The second method we consider is a recent variant of minimum degree, called Approximate Minimum Degree (AMD) [1]. AMD further reduces runtime by computing an inexpensive upper bound on a vertex's degree rather than the true degree. Another recently proposed variant of minimum degree, Approximate Minimum local Fill (AMF) [29], improves on minimum degree by modifying the strategy used to select vertices for elimination. The method uses a rough approximation of the fill that would be generated by eliminating a vertex rather than using the vertex degree. The runtime of AMF ordering is only slightly higher than that of AMD ordering.

2.2 Nested dissection ordering

Nested dissection takes a very different approach to ordering a sparse matrix. Whereas minimum degree considers very local information about the graph G (vertex degree), nested dissection takes a more global view. Specifically, the nested dissection algorithm performs the reordering on G as follows:

- Find a vertex separator S in G (a set of vertices whose removal leaves a disconnected graph).
- Remove the vertices in S from G (and order them after the remaining vertices in G).
- Perform nested dissection on the sub-graphs that remain.

By ordering S after the remaining, disconnected sub-graphs, the nested dissection algorithm guarantees that the sub-graphs are disconnected in the factor matrix L, thus limiting potential fill. The vertices in S generally form a dense sub-matrix in L, so the quality of the ordering clearly depends on the size of S.

As mentioned earlier, several methods are available for finding vertex separators S in G. This paper concentrates on a particular class of these methods, multi-level variants of Fiduccia-Mattheyses graph partitioning, that has proven to be quite effective for structural analysis and computational fluid dynamics matrices [7, 14, 17]. The algorithms we employ here are described more fully in [15].

Multi-level methods for graph partitioning are based on the following framework. The original graph is first *coarsened*, by forming a sequence of graphs, where each graph in the sequence ideally has half the number of vertices as the previous one. Each graph in the sequence is a coarser "approximation" of the previous graph, typically obtained by merging pairs of adjacent vertices in the large graph into single vertices in the coarsened graph. The coarsest graph is then partitioned into two sets. This partitioning is then *projected* onto the next larger graph. The projected partitioning is refined in the larger graph, using some local improvement heuristic. The resulting partitioning is then projected onto the next larger graph. The process continues until a partitioning on the original, uncoarsened graph is found.

The local improvement heuristic used on the coarsened graphs in the multi-level method is typically a linear-time variant of the Fiduccia-Mattheyses graph partitioning algorithm [8, 18]. We refer the reader to the relevant papers for details. As originally stated, the Fiduccia-Mattheyses graph partitioning heuristic computes edge separators; it partitions the vertices of G into two sets, C and D, attempting to minimize the number of edges between the sets. Recall that the nested dissection method for ordering sparse matrices requires vertex separators — it requires three sets of vertices, S, C, and D, such that C and D have no edges between them. A vertex separator can trivially be derived from an edge separator by choosing S to be the set of all vertices in C that are adjacent to vertices in D (or vice versa). Alternatively, one can use bipartite graph matching techniques to find the minimum size vertex separator from among all vertices incident to separator edges [25, 28]. The cost of computing the minimum cost separator from among these vertices is quite small.

One obvious limitation of an edge separator approach to finding vertex separators is that it optimizes the wrong objective function. The heuristic tries to minimize the number of edges between C and D, while the quality of the nested dissection ordering depends on the number of vertices in the derived vertex separator S. An alternative to the use of edge separators is an approach that finds vertex separators directly [2, 19]. One can create a multi-level vertex separator method [15] that is entirely analogous to the multi-level edge separator approach, except that it maintains three sets of vertices at each level of the graph coarsening: S, C, and D. The local improvement heuristic then attempts to directly minimize the size of the separator S at each level.

In our nested dissection implementation, we recursively divide the graph until the remaining sub-graphs contain fewer than 1/32 of the original vertices. At this point, the sub-graphs are ordered using AMD. This hybrid approach takes advantage of the ability of minimum degree methods to quickly order small domains very effectively.

Nested dissection can be hybridized with a minimum degree method in another way [3, 15]. Once all sub-graphs containing fewer than 1/32 of the original vertices are reordered and eliminated, minimum degree ordering can be used to reorder the remaining separator vertices. This has the advantage of removing the implicit and often incorrect assumption in nested dissection that a pure recursive subdivision of the problem is best. This paper presents results from two different nested dissection approaches, one that uses the original recursive ordering of separator vertices and another that reorders these separator vertices using AMF.

The results in [15] indicate that nested dissection methods based on vertex separators are more effective than either multi-level edge separator nested dissection methods or minimum degree methods for structural analysis and computational fluid dynamics problems. It is not at all clear, however, how such methods would behave for linear programming problems. Nested dissection is based on geometric intuitions that might not necessarily hold for the highly irregular problems found in linear programming. In particular, the graphs associated with LP matrices may not have the small separators which motivate nested dissection. There has been some previous work applying nested dissection using edge separator methods to relatively small LP models with mixed results [16, 17]. We now consider the behavior of a variety of ordering methods on a suite of large-scale linear programming problems.

3 Methodology

Our study looks at 22 large-scale linear programming problems. The problem suite includes four widely studied problems from NETLIB [9] (PILOT, PILOT87, DFL001, and PDS-20), and a variety of problems from customer applications. This test set includes problems from telecommunications, the electronics industry, satellite resource allocation, FAA aircraft slot assignment, and airline fleet models from five different airlines. This suite was chosen to represent a cross-section of large-scale LP problems available to us, and all were chosen before any results in this paper were generated. While airline fleet models may appear over-represented in this problem set, in our experience this application generates a significant fraction of the large-scale LP models currently being solved.

All matrices were extracted from CPLEX 4.0 (a commercial math programming package) after CPLEX presolve was applied to the problems. Table 1 shows relevant information about these problems, including the number of rows in the matrix, the number of non-zero values in the lower triangle of $A\Theta A^T$ (in thousands), the number of non-zero values in the lower triangle of L after applying Liu's multiple minimum degree ordering [20] (in thousands), and the number of floating-point operations required to perform the factorization (in millions) after MMD ordering.

Table 1: Statistics about test matrices.

Matrix	Rows	NZ in $A\Theta A^T$ (10^3)	NZ in L (10^3)	Operations to factor (10^6)
A15	6330	192	8197	17938
DFL001	3965	42	1297	1078
FAA	103348	1234	5928	6741
TELECOM	22919	478	2593	1025
GISMONDI	11884	377	45777	278105
ELECTRONICS	16314	4524	5571	2602
RAILROAD	72291	3738	13471	6841
PDS-20	28135	158	5916	7007
PILOT	1275	58	195	51
PILOT87	1890	116	421	179
DISTRIBUTION	22972	12425	49144	169254
MULTICOM	30259	804	197598	2283557
SATELLITE	4667	504	2388	1525
FLEET1	20003	169	11294	30239
FLEET2	60589	865	134615	975363
FLEET3	16678	289	5411	7717
FLEET4	16704	165	5394	7379
FLEET5	2256	84	1562	1761
FLEET6	16970	190	6837	11122
FLEET7	13946	254	3732	4184
FLEET8	32759	249	32196	124470
FLEET12	18835	196	4647	5560

We order our suite of matrices using the five different codes. For our MMD results, we use Liu's implementation [20]. For AMD, we use the implementation by Amestoy, Davis, and Duff [1]. For AMF, we use our own code. We should note that this implementation is built on top of an AMD code that is less efficient than the one by Amestoy, Davis, and Duff. The multi-level edge Fiduccia-Mattheyses (EFM) nested dissection method is built on top of the Chaco graph partitioning package [13]. The multi-level vertex Fiduccia-Mattheyses (VFM) nested dissection method was implemented by Hendrickson and Rothberg [15] in a software tool called BEND. All experiments described in this paper were performed on a Silicon Graphics R8000 Power Challenge system.

3.1 Floating-point operation counts

Table 2 shows the number of floating-point operations required to factor the $A\Theta A^T$ matrices for each matrix in our suite. Note that all results are expressed relative to the operation count for MMD. Since MMD is the most widely used method for ordering sparse matrices, such ratios hopefully make the results easier to interpret. Note that actual floating-point operation counts can be recovered by multiplying these ratios by the MMD operation counts in Table 1.

As can be seen from the table, AMD gives roughly comparable orderings to MMD. One can observe minor variations from problem to problem, but this is to be expected with any ordering heuristic. These results are consistent with those for structural analysis matrices [1]. It can also be observed from the table that AMF produces roughly 14% better orderings than MMD. The vertex selection criteria of AMF is clearly more effective than that of MMD and AMD.

Table 2: Floating-point operations for factorization (relative to MMD).

Matrix	AMD	AMF	Edge FM	Vertex FM	VFM+ AMF
A15	1.34	1.11	1.15	0.97	0.92
DFL001	0.94	0.89	0.89	0.64	0.65
FAA	1.04	0.97	7.20	2.19	1.06
TELECOM	0.72	0.55	0.62	0.44	0.35
GISMONDI	1.01	0.93	0.41	0.42	0.44
ELECTRONICS	1.01	0.80	2.39	0.74	0.75
RAILROAD	0.92	0.79	3.37	0.78	0.77
PDS-20	1.03	1.17	0.92	0.41	0.27
PILOT	0.92	0.76	1.48	0.74	0.65
PILOT87	1.02	1.00	1.07	0.90	0.90
DISTRIBUTION	1.28	0.51	0.24	0.21	0.20
MULTICOM	0.90	0.74	0.05	0.10	0.05
SATELLITE	0.94	1.32	1.67	1.18	1.07
FLEET1	1.12	0.96	0.91	0.66	0.51
FLEET2	1.09	0.92	0.36	0.23	0.24
FLEET3	0.94	0.71	1.99	0.45	0.49
FLEET4	0.83	0.71	2.50	0.39	0.38
FLEET5	1.01	1.00	1.64	1.04	1.03
FLEET6	1.07	0.97	0.75	0.41	0.34
FLEET7	1.03	0.84	1.99	0.47	0.44
FLEET8	1.00	0.86	0.97	0.58	0.55
FLEET12	0.97	0.81	1.90	0.51	0.45
Geom Mean	1.00	0.86	1.05	0.54	0.48

Moving to the nested dissection results, we find that the multi-level edge Fiduccia-Mattheyses method require 5% more floating-point operations than MMD on average, although the variation is quite high. On problems DISTRIBUTION and MULTICOM, for example, this method reduces operation counts by factors of roughly 4 and 20, respectively. For problems FAA and RAILROAD, however, the method increases operation counts by factors of roughly 7 and 3. Overall, the method increases operation counts for 12 problems and reduces them for 10.

In contrast, the multi-level vertex Fiduccia-Mattheyses method produces consistently better orderings than MMD. Of the 22 problems, it only produces significantly worse orderings for two (FAA and SATELLITE). Clearly, the use of a vertex separator approach rather than an edge separator approach significantly improves the behavior of the method. Overall, the vertex FM method reduces operation counts by 46%.

Performing AMF on the separators further reduces the operation counts for 17 of the 22 matrices. The previously troublesome FAA matrix is improved dramatically. Overall, the orderings produced this way require less than half as many floating point operations as those produced by MMD.

An obvious question at this point is why a nested dissection approach significantly outperforms minimum degree for most of these matrices. While the answer will of course depend on the particular matrix, some insight can be gained from considering the matrices that arise in multi-stage stochastic linear programming, where a domain-specific nested dissection ordering outperformed MMD by a factor of 1000 [5]. The matrices from this problem domain all contain a family of very small separators. Eliminating these separator vertices last allows the problem to be trivially decomposed into many small sub-problems. Unfortunately, no degree cues are available to indicate that these separator vertices should be eliminated last. MMD eliminates these vertices early in the process, thus destroying the natural separator structure. While the multi-stage stochastic linear programming problems were an extreme case, we suspect that many of the matrices in our test suite contain small separators that MMD eliminates early because they lack obvious degree cues.

A second obvious question is why the multi-level vertex separator method is vastly superior to the multi-level edge separator approach. Similar comparisons for matrices arising in structural analysis and computational fluid dynamics indicate that a vertex approach is preferable [15], but the difference in that setting is much smaller. Unlike the matrices from those scientific computing applications, LP matrices tend to have highly variable degrees. If high-degree vertices are in the best vertex separator, neighboring edge separators will generally have a large number of edges. Consequently, an edge-oriented approach will find worse vertex separators than a vertex-oriented method.

The reader should note that while the factorization is the dominant cost in the interior point method, it is not the only cost. Thus, the 52% reduction in operation counts would translate to a somewhat smaller reduction in the execution time of the method as a whole.

3.2 Factor storage

The interior point method is often quite memory-intensive, and the majority of this memory is typically used to hold the factor matrix L. Table 3 shows the number of non-zero values in the factor matrices, again relative to MMD, for the various ordering strategies. On average, AMD produces roughly the same number

Table 3: Non-zero values in factor matrix L (relative to MMD).

Matrix	AMD	AMF	Edge FM	Vertex FM	VFM+ AMF
A15	1.14	1.05	1.09	1.00	0.97
DFL001	0.98	0.95	1.03	0.86	0.86
FAA	1.01	0.99	3.07	1.62	1.12
TELECOM	0.88	0.80	0.90	0.76	0.70
GISMONDI	1.00	0.97	0.68	0.68	0.69
ELECTRONICS	1.00	0.93	1.41	0.91	0.91
RAILROAD	0.99	0.94	1.51	0.95	0.94
PDS-20	1.02	1.05	1.05	0.68	0.59
PILOT	0.97	0.90	1.18	0.90	0.84
PILOT87	1.00	1.01	1.10	0.99	0.98
DISTRIBUTION	1.12	0.76	0.55	0.51	0.50
MULTICOM	0.96	0.87	0.23	0.29	0.23
SATELLITE	0.97	1.14	1.28	1.07	1.04
FLEET1	1.06	0.98	1.10	0.86	0.76
FLEET2	1.04	0.94	0.63	0.48	0.49
FLEET3	0.97	0.85	1.48	0.72	0.72
FLEET4	0.91	0.85	1.72	0.66	0.64
FLEET5	1.00	1.00	1.39	1.05	1.04
FLEET6	1.02	0.97	0.91	0.69	0.62
FLEET7	1.02	0.92	1.48	0.73	0.71
FLEET8	0.99	0.92	1.07	0.77	0.73
FLEET12	0.98	0.90	1.45	0.75	0.70
Geom Mean	1.00	0.94	1.08	0.77	0.73

of non-zero values as MMD, AMF produces 6% fewer, while the multi-level edge FM method produces 8% more. The multi-level vertex FM method produces 23% fewer, and after reordering the separators with AMF the reduction becomes 27%.

3.3 Ordering runtimes

In an interior point method, the time required to compute an ordering using MMD is typically not a large fraction of the total solution time. Recall that the non-zero structure of the matrix does not change from iteration to iteration, so the ordering only has to be performed once. In contrast, the linear system $A\Theta A^T$ must be solved 30–60 times. One important case where ordering times become important, however, is when the interior point method is performed on a parallel machine [6, 16, 23]. The computations associated with the iterations of the method, including the factorization, can be readily parallelized, while ordering, in general, is more difficult to parallelize. We should add that nested dissection orderings are often advantageous for parallel factorization (see [26], for example). Table 4 shows runtimes for the various ordering methods, again relative to MMD. The table shows that all of the alternatives we have considered here are significantly

Table 4: Ordering runtime (relative to MMD).

Matrix	AMD	AMF	Edge FM	Vertex FM	VFM+ AMF
A15	0.04	0.09	0.09	0.09	0.11
DFL001	0.11	0.20	0.35	0.35	0.47
FAA	0.06	0.17	0.21	0.32	0.42
TELECOM	0.14	0.23	0.43	0.44	0.47
GISMONDI	0.08	0.15	0.39	0.52	0.58
ELECTRONICS	0.11	0.59	0.32	0.22	0.63
RAILROAD	0.11	0.13	0.19	0.19	0.24
PDS-20	0.01	0.01	0.02	0.01	0.02
PILOT	0.16	0.42	0.69	0.79	0.92
PILOT87	0.11	0.16	0.46	0.54	0.63
DISTRIBUTION	0.06	0.08	0.25	0.26	0.26
MULTICOM	0.03	0.06	0.07	0.08	0.10
SATELLITE	0.07	0.18	2.36	2.72	2.74
FLEET1	0.08	0.05	0.14	0.15	0.17
FLEET2	0.03	0.04	0.11	0.10	0.12
FLEET3	0.10	0.14	0.25	0.28	0.31
FLEET4	0.09	0.10	0.16	0.18	0.20
FLEET5	0.04	0.08	0.21	0.15	0.20
FLEET6	0.16	0.15	0.39	0.44	0.47
FLEET7	0.10	0.14	0.26	0.27	0.31
FLEET8	0.05	0.08	0.14	0.15	0.20
FLEET12	0.19	0.06	0.34	0.33	0.32
Geom Mean	0.07	0.11	0.23	0.24	0.29

less expensive than MMD. AMD reduces ordering times by an average factor of fourteen for this suite of large problems. AMF reduces ordering times by a factor of nine. Interestingly, the nested dissection methods reduce ordering times by roughly a factor of four, even though these same methods significantly *increase* ordering times over MMD for structural analysis and CFD problems [15].

Upon further investigation, we discovered several properties in these linear programming matrices that can cause large runtimes in MMD when compared with structural analysis and CFD matrices. The first is that linear programming problems often have many vertices with very high vertex degrees. In problem FLEET12, for example, the matrix contained 18835 vertices, and one vertex had degree 18699. Nearly every elimination step in the MMD algorithm removes a neighbor of this high-degree vertex, so the degree of this high degree vertex must be updated. The degree update requires a very expensive traversal of the full adjacency list. FLEET12 was an extreme example, but we observed many high-degree vertices in the matrices in our test set. In contrast, vertex degrees in structural analysis and CFD problems are generally quite low and relatively uniform.

Another reason for large MMD runtimes for these problems is that multiple elimination is less effective than it is for structural analysis matrices. Recall that MMD reduces runtimes by simultaneously eliminating multiple independent nodes of minimum degree. For LP matrices, we found that the higher variability in node degree limits the number of vertices of minimum degree. We also found that the vertices in LP matrices are more interconnected than those in structural analysis matrices, which reduces the number of these minimum degree vertices that are independent. The result is that fewer vertices are eliminated in each multiple elimination stage for linear programming matrices than for structural analysis matrices. This increases the cost of the MMD algorithm.

4 Conclusions

This paper has considered the question of how linear programming matrices should be ordered prior to factorization. The most widely used reordering scheme for this problem is multiple minimum degree (MMD) but our results indicate that this is no longer the appropriate choice. Approximate minimum degree produces orderings of equivalent quality while running more than an order of magnitude faster. Approximate minimum local fill reduces floating point operations by 14%, while still running 9 times faster than MMD. And a nested dissection approach based on a multi-level vertex Fiduccia-Mattheyses partitioning scheme reduces floating-point operations by more than a factor of 2, reduces storage requirements by 27%, and reduces ordering runtimes by roughly a factor of four.

Acknowledgements

The authors would like to thank Bob Bixby and Irv Lustig of CPLEX Optimization for providing access to many of the large linear programming models studied in this paper.

References

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

[2] Ashcraft, C., and Liu, J., *A partition improvement algorithm for generalized nested dissection*, Technical Report BCSTECH-94-020, Boeing Computer Services, 1994.

[3] Ashcraft, C., and Liu, J., *Robust ordering of sparse matrices using multisection*, Technical Report ISSTECH-96-002, Boeing Information and Support Services, 1996.

[4] Ashcraft, C., and Liu, J., *Using domain decomposition to find graph bisectors*, Technical Report ISSTECH-95-024, Boeing Information and Support Services, 1995.

[5] Berger, A., Mulvey, J., Rothberg, E., and Vanderbei, R., *Solving multistage stochastic programs using tree dissection*, Technical Report SOR-95-07, Dept. of Statistics and Operations Research, Princeton University, June, 1995.

[6] Bisseling, R., Doup, T., and Loyens, L., "A parallel interior point algorithm for linear programming on a network of transputers", *Annals of Operations Research*, 43: 51–86, 1993.

[7] Bui, T, and Jones, C., "A heuristic for reducing fill in sparse matrix factorization", *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 1993.

[8] Fiduccia, C., and Mattheyses, R., "A linear time heuristic for improving network partitions", in *Proceedings of the 19th IEEE Design Automation Conference*, pp. 175-181, 1982.

[9] Gay, D., "Electronic mail distribution of linear programming test problems", *Mathematical Programming Society (COAL) Newsletter*, 1988.

[10] George, A., and Liu, J., "An automated nested dissection algorithm for irregular finite element problems", *SIAM J. Numer. Anal.*, 15: 1053-1069, 1978.

[11] George, A., and Liu, J., *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, 1981.

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

[13] Hendrickson, B., and Leland, R., *The Chaco User's Guide: Version 2.0*, Sandia National Laboratories Technical Report SAND94-2692, October, 1994.

[14] Hendrickson, B., and Leland, R., "A multilevel algorithm for partitioning graphs", in *Proceedings in Supercomputing '95*, November, 1995.

[15] Hendrickson, B., and Rothberg, E., *Improving the runtime and quality of nested dissection ordering*, Technical Report SAND96-0868J, Sandia National Laboratories, March, 1996.

[16] Karypis, G., Gupta, A., and Kumar, V., "A parallel formulation of interior point algorithms", in *Proceedings of Supercomputing '94*, pp. 204–213, November, 1994.

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

[18] Kernighan, B., and Lin, S., "An efficient heuristic procedure for partitioning graphs", *Bell System Technical Journal*, 29: 291-307, 1978.

[19] Liu, J., "A graph partitioning algorithm by node separators", *ACM Trans. Math. Software*, 15(3): 198-219, 1989.

[20] Liu, J., "Modification of the minimum degree algorithm by multiple elimination", *ACM Trans. Math. Software*, 11, pp. 141-153, 1985.

[21] Lustig, I., Marsten, R., and Shanno, D., "Computational experience with a primal-dual interior point method for linear programming", *Linear Algebra and Its Applications*, 152: 191-222, 1991.

[22] Lustig, I., Marsten, R., and Shanno, D., "Interior point methods for linear programming: computational state of the art", *ORSA Journal on Computing*, 6(1): 1-14, 1994.

[23] Lustig, I., and Rothberg, E. "Gigaflops in linear programming", to appear in *OR Letters*.

[24] Markowitz, H., "The elimination form of the inverse and its application to linear programming", *Management Science*, 3: 255-269, 1957.

[25] Pothen, A., and Fan, C., "Computing the block triangular form of a sparse matrix", *ACM Trans. Math. Soft.*, 16(4); 303-324, December, 1990.

[26] Pothen, A., Rothberg, E., Simon, H., and Wang, L., "Parallel sparse Cholesky factorization with spectral nested dissection ordering", in *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pp. 418-422, June, 1994.

[27] Pothen, A., Simon, H., and Liou, K., "Partitioning sparse matrices with eigenvectors of graphs", *SIAM Journal of Matrix Analysis and Applications*, 11(3): 430-452, 1990.

[28] Pothen, A., Simon, H., Wang, L., and Barnard, S., "Towards a fast implementation of spectral nested dissection", *Proceedings of Supercomputing '92*, pp. 42-51, 1992.

[29] Rothberg, E., "Ordering sparse matrices using approximate minimum local fill", Silicon Graphics manuscript, submitted for publication, April, 1996.

[30] Tinney, W., "Comments on using sparsity techniques for power system problems", in *Sparse Matrix Proceedings*, IBM Research Report RAI 3-12-69, 1969.