GRAPH PARTITIONING ALGORITHMS WITH APPLICATIONS TO SCIENTIFIC COMPUTING

Alex Pothen[1]

Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
pothen@cs.odu.edu

ICASE, MS 132C
NASA Langley Research Center
Hampton, VA 23681-0001
pothen@icase.edu

ABSTRACT

Identifying the parallelism in a problem by partitioning its data and tasks among the processors of a parallel computer is a fundamental issue in parallel computing. This problem can be modeled as a graph partitioning problem in which the vertices of a graph are divided into a specified number of subsets such that few edges join two vertices in different subsets. Several new graph partitioning algorithms have been developed in the past few years, and we survey some of this activity.

We describe the terminology associated with graph partitioning, the complexity of computing good separators, and graphs that have good separators. We then discuss early algorithms for graph partitioning, followed by three new algorithms based on geometric, algebraic, and multilevel ideas. The algebraic algorithm relies on an eigenvector of a Laplacian matrix associated with the graph to compute the partition. The algebraic algorithm is justified by formulating graph partitioning as a quadratic assignment problem. We list several papers that describe applications of graph partitioning to parallel scientific computing and other applications. Finally we discuss the application of graph partitioning to obtain nested dissection orderings for solving sparse linear systems of equations in parallel.

[1]This work was supported by National Science Foundation grants CCR-9412698 and DMS-9505110, by U. S. Department of Energy grant DE-FG05-94ER25216, and by the National Aeronautics and Space Administration under NASA Contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

1 Introduction

We survey recent progress in the design of algorithms and software for partitioning graphs. New partitioning algorithms that rely on combinatorial, algebraic, and geometric ideas have been created and carefully implemented in the last few years. These contributions make it possible to compute high quality partitions of large graphs (say, graphs with about one hundred thousand vertices) in a few minutes on a workstation. Much progress has been made to date, and continues to be made at this time, in the design of new algorithms and in understanding their behavior. This paper is a snapshot of a dynamic area of research from the personal perspective of one of the participants.

Graph partitioning plays a fundamental role in parallel computing by identifying the concurrency in a given problem when the computation can be modeled by a graph. A partition of the graph into subgraphs leads to a decomposition of the data and/or tasks associated with a computational problem and the subgraphs can then be mapped to the processors of a multiprocessor.

Graph partitioning also has an important role to play in the design of many serial algorithms by means of the divide and conquer paradigm. Two important examples of this algorithmic paradigm are in the solution of partial differential equations (PDEs) by domain decomposition and in the computation of nested dissection orderings for solving sparse linear systems of equations. Other applications include circuit partitioning and layout, VLSI design, and Computer-Aided-Design. One application we consider is finite element discretizations of PDEs, so that the graphs in this context are finite element meshes.

Two objectives are usually stated in the partitioning problem: partition a given graph into a specified number of subgraphs such that the subgraphs have roughly equal numbers of vertices, and few edges join different subgraphs to each other (these edges are "cut" by the partition). In the context of parallel computations, the size of a subgraph determines the computational work that a processor has to perform, and the number of cut edges is a measure of the communication volume in the algorithm. In a serial algorithm, equal-sized subgraphs lead to the lowest worst-case running time, and the number of cut edges (or the number of the vertices these edges are incident on) measures the cost of combining the partial solutions to compute a solution.

More general objective functions may need to be considered for many problems. The work associated with a subgraph may be modeled more accurately by attaching a weight to each vertex, and then equi-partitioning the weights. The "communication" costs in the algorithm might be modeled more accurately by the number of subgraphs a given subgraph is connected to, or the number of "boundary" vertices, or similar variants. In addition, the shape of the subgraphs (e.g., the aspect ratios) may be an important parameter in some algorithms (e.g., the convergence of a domain decomposition method). In some applications, it may be essential to partition into connected subgraphs.

Nevertheless, we focus primarily on the archetypical problem of partitioning a graph into subgraphs with roughly equal numbers of vertices to minimize the number of cut edges. Most of the algorithms that we describe can be adapted to include more general objective functions. Even the ones that cannot be so adapted can be used to obtain an initial partition; this partition can then be improved iteratively in a second phase using a "local" algorithm that moves vertices among the subgraphs to optimize the partition with respect to the more general objective function.

In Figure 1, we show the partition of a mesh into eight subgraphs computed by the spectral partitioning algorithm. This mesh was generated using the SCOREC mesh generator at Rensselaer Polytechnic Institute [95]; it is the dual of a surface mesh of a Commanche helicopter, and has $7,920$ vertices and $11,880$ edges. The figure shows that each subgraph in the partitioned mesh is connected, and it appears to the eye as a "good" partition of the mesh. Quantitative measures such as the number of edges cut by the partition show that the spectral algorithm does generate partitions of good quality for many finite element meshes.

The rest of this paper is organized as follows. We describe the terminology of graph partitioning, and discuss known results about the quality of partitions in Section 2. Some early partitioning algorithms are described in Section 3. In the next few sections we proceed to discuss the new partitioning algorithms: the spectral algorithm in Section 4, a geometric partitioning algorithm in Section 5, and a multilevel algorithm in Section 6. Section 7 describes a formulation of the partitioning problem as a mathematical programming problem (a quadratic assignment problem) that leads to a justification of the spectral algorithm. Papers dealing with several applications

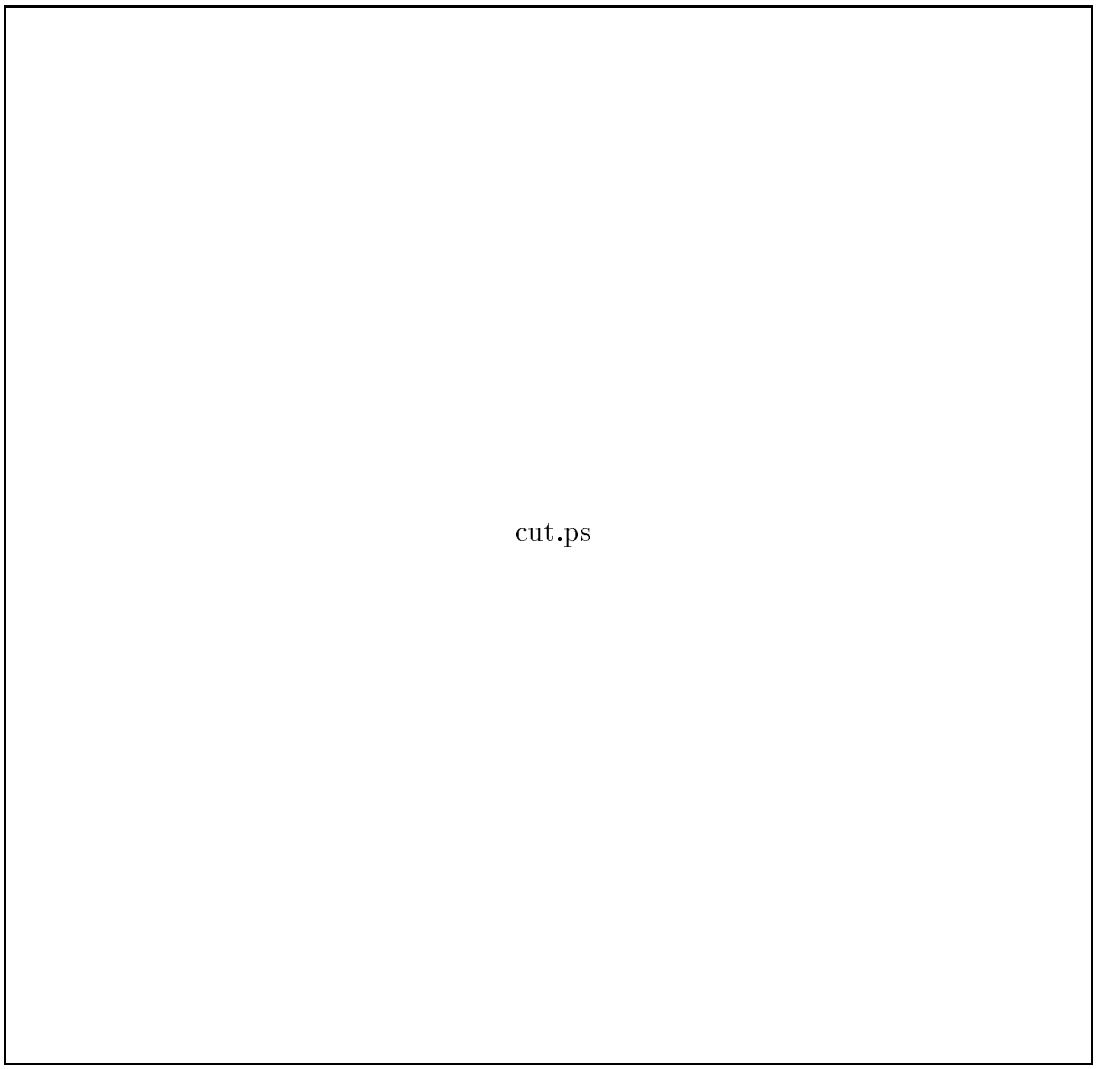

Figure 1: The Commanche mesh partitioned into 8 subgraphs with the spectral partitioning algorithm.

are listed in Section 8; one application, the computation of nested dissection orderings, is discussed in detail.

We now briefly consider some of the other partitioning algorithms not described in detail in this survey.

Simulated annealing has been used to partition graphs [61], but computational results in [32] show that it provides edge separators of inferior quality and requires much greater computational times when compared with the spectral partitioning algorithm. Genetic algorithms have also been recently designed for the partitioning problem [16], and their performance is comparable to that of simulated annealing. Another interesting partitioning approach is by means of the Peano-Hilbert curve [83]. The elements in a finite element mesh are ordered in a linear order by mapping the centroids of the elements to a Peano-Hilbert curve, and an element is chosen such that all elements numbered lower belong to the first part, and the rest to the second part. The element to partition can be chosen to reduce either the number of edges cut or a more general objective function that combines load balance and edge separator size.

A very recent separator algorithm due to Ashcraft and Liu [5], computes a separator in a novel way: They first partition the graph into several subgraphs to obtain a "multisector" from the vertices on the partition boundaries, combine the subgraphs to obtain a partition into two subgraphs, then use graph matching techniques to identify a small separator.

Notation. We will denote a graph G by means of its set of vertices V, and the set of edges E. An edge in E is a pair of vertices (u, v); the vertices u and v are the endpoints of the edge. The number of elements in a set S will be indicated by $|S|$. Often, we will consider graphs on n vertices, and thus $|V| = n$. Graph-theoretic concepts not defined here could be found in any introductory book in graph theory (e.g.,Bollobás [11]), and similarly, concepts in linear algebra in a book on matrix computations (e.g., Golub and Van Loan[46]).

2 Background and graphs with good separators

A *partition* of a connected graph $G = (V, E)$ is a division of its vertices into two sets A and B. The set of edges joining vertices in A to vertices in B is an *edge separator* that we shall denote by $\delta(A, B)$;

the removal of these edges disconnects the graph into two or more connected components. In applications such as domain decomposition, the set of vertices A would be mapped to one set of processors and the set of vertices B to another, and $|\delta(A,B)|$ is a measure of the volume of communication necessary between the two groups of processors. Hence one goal in partitioning a graph in these applications is to minimize the number of edges cut by the partition so as to keep communication costs in the algorithm small.

A second goal would be to balance the computational work (load) between the two sets of processors. This is achieved by prescribing the number of vertices in A and B to within a tolerance. If A and B are equal in size, then the partition is called a *bisection* and $|\delta(A,B)|$ is the bisection width.

In other applications such as nested dissection, a *vertex separator* is desired; this is a set of vertices S whose removal disconnects the graph into two parts with no edge joining a vertex in one part to a vertex in the other. Here the two goals are that the separator should have a small number of vertices, and as before, that the two parts should not differ by too many vertices.

Given an edge separator, we can compute a vertex separator from it by choosing the endpoints of $|\delta(A,B)|$ belonging to either A or B. However, consider the bipartite graph induced by $\delta(A,B)$ (i.e., the graph consisting of these edges and their endpoints). Any *vertex cover* of this induced graph (a set of vertices such that every edge has at least one endpoint in the set) will serve as a vertex separator of the graph G. It is possible to find a *minimum cover*, a cover with the fewest vertices, by means of a maximum matching.

Most variants of the graph partitioning problem are NP-hard, i.e., it is unlikely that vertex separators or edge separators of minimum cardinality can be computed efficiently (in polynomial time) for general classes of graphs. Worse, Bui and Jones [14] have shown that it is NP-hard to find approximately optimal vertex and edge separators, even in graphs with maximum degree three. More precisely, it is NP-hard to compute an approximately optimal separator that contains at most $n^{(1/2)-\epsilon}$ vertices more than an optimal separator, for any positive ϵ.

However, it is possible to find asymptotically small, though not necessarily optimal, separators for large classes of graphs that arise in practical situations such as finite element and finite difference solutions of partial differential equations. We state several vertex

separator size results in the following paragraphs, and similar results hold for edge separators as well.

The oldest separator result is for trees, and is due to Jordan in 1869: Every tree has a single vertex that separates it into two parts, with no part containing more than two-thirds of the vertices. Lipton and Tarjan [75] showed that every planar graph has a vertex separator of size at most $\mathcal{O}(\sqrt{n})$ separating it into two parts with no part having more than two-thirds the number of vertices. These results extend to two-dimensional finite element graphs. Vertex separators that are bounded by $\mathcal{O}(\sqrt{n})$ exist also for graphs with bounded genus and for graphs with certain excluded minors.

Miller, Teng, Thurston, and Vavasis [79] have recently extended these results to a more general class of graphs embedded in d dimensions To define this class of overlap graphs, we begin with a set $\{B_1, \ldots, B_n\}$ of closed balls in $\Re^d$. A parameter k determines how closely these balls can intersect: if any point $p \in \Re^d$ lies strictly within at most k balls, then the set of balls is called a k-ply neighborhood system. We obtain an overlap graph by creating a vertex for each ball, and joining two vertices with an edge if the corresponding balls intersect.

It is possible to include another parameter $\alpha \geq 1$, which determines by how much the radius of each ball can be enlarged to make two balls intersect, and thus obtain a more general construction of overlap graphs. If a ball B_i centered at p_i has radius r_i, then the expanded ball αB_i is the ball with the same center but with radius $\alpha\, r_i$. An overlap graph (with parameters α and k) is a k-ply neighborhood system where a vertex represents each ball, and where an edge joins two vertices if the corresponding balls intersect when the smaller of the balls is expanded by a factor α.

Overlap graphs can be embedded in space by locating a vertex corresponding to a ball B_i at the point p_i, the center of the ball. Planar graphs, two-dimensional meshes, and three dimensional meshes of bounded aspect ratio can be represented by overlap graphs for suitable choices of the parameters α and k.

Miller et al. [79] proved that an (α, k)-overlap graph has a vertex separator of size $\mathcal{O}(\alpha k^{1/d} n^{1-\frac{1}{d}})$; the larger of the two parts obtained by removing the separator has at most $(d+1)/(d+2)$ vertices. The geometric algorithm for computing this separator is described in Section 5.

It should be reiterated that these asymptotic bounds on the size

of vertex separators are not necessarily optimal. For instance, Bui, Fukuyama, and Jones have shown that finding the smallest vertex separator in a planar graph is NP-complete [13].

It is possible to obtain lower bounds on the sizes of vertex separators and edge separators by formulating the partition problem as a mathematical programming problem, and relaxing the integrality constraints in the problem; such bounds are described in Section 7. Lower bounds on edge separators by this technique were first obtained by Donath and Hoffman [29]; the strongest current bounds are due to Rendl and Wolkowicz [92]. These bounds potentially could be used to state how close a computed edge separator is to an optimal separator by finding the difference between the size of the computed separator and the lower bound on the separator. Unfortunately the lower bounds on the edge separator are too weak for this purpose. However, these techniques yield surprisingly tight lower bounds (to within a few percent, for large classes of graphs including many finite element meshes) for a vertex numbering problem on graphs called the 2-sum problem [42].

3 Early partitioning algorithms

In this section we briefly describe three of the earlier algorithms for partitioning graphs: the Kernighan-Lin algorithm, a partitioning algorithm based on computing level structures, and an inertial algorithm.

3.1 The Kernighan-Lin algorithm

This is one of the earliest algorithms proposed for partitioning graphs (published in 1970), and variants are still in use today. Kernighan and Lin [66] were motivated by the problem of partitioning electronic circuits onto cards: each card contains a subset of nodes, and the objective is to minimize the number of connections between nodes assigned to different cards.

A graph model of the problem is to partition the vertices of a graph into k subsets so as to minimize the number of edges joining vertices in different subsets. We assume for simplicity that the number of subsets k and the number of nodes in the subsets are specified.

The algorithm begins with an initial partition into two sets, obtained by a random partition or from some initial information avail-

able about the problem. At each iteration the algorithm swaps subsets consisting of equal numbers of vertices between the two sets to reduce the number of edges joining the two sets. The algorithm terminates when it is no longer possible to reduce the number of edges by swapping subsets, or when a specified number of swaps have been made. We can partition a graph into more than two subsets by recursively applying the bipartition algorithm.

How does the algorithm choose subsets of vertices to swap? This choice is based on the fundamental concept of the *gain* associated with moving a vertex v belonging to one set A to the other set B. This gain is the net reduction in the number of edges cut by the partition. Edges joining v to vertices in B are cut in the initial partition, whereas after the swap, edges joining v to A are cut by the new partition. Thus the gain in moving the vertex v from the set A to the set B is the difference between the number of edges joining v to vertices in B and the number of edges joining v to vertices in A. If this gain is positive, then moving the vertex v from A to B will reduce the number of edges cut. We could then find a vertex with positive gain to move from B to A to restore the sizes of the two sets. We could continue in this manner, exchanging vertices with positive gains, and thereby reducing the number of cut edges.

An important idea in the Kernighan-Lin algorithm is to continue to move vertices with negative gains for a preset number of steps, in the hope that such a sequence of moves might create vertices with positive gains later on, and thus reduce the number of edges cut overall. This enables the algorithm to climb out of local minima in partition space. If the best partition found so far is recorded, then when a sequence of moves with negative gains does not improve the cut size, we restore the current partition to the best partition obtained during the current iteration of the algorithm.

The algorithm described by Kernighan and Lin for the bipartition problem requires $\mathcal{O}(n^2 \log n)$ time, if we assume that only a constant number of subset-swaps are performed at each iteration of the algorithm. An important practical advance was made by Fiduccia and Mattheyses who implemented an iteration of the Kernighan-Lin algorithm to run in $\mathcal{O}(|E|)$ time [35] . Fiduccia and Mattheyses described how to compute and update the gains of the vertices that are candidates to be moved during an iteration of the algorithm. In their efficient implementation, the gains associated with the vertices are sorted by value, and the moves associated with each gain value are

stored in a doubly linked list. Choosing a move with the highest gain value involves finding a nonempty list with the highest gain, while updating the gain of a vertex is accomplished in constant time by deleting it from one doubly linked list and inserting it into another.

The algorithm was generalized to partition a graph directly into four subsets by Suaris and Kedem [98]; it is also possible to partition the graph into an arbitrary number of sets directly. However, the run-time and storage costs of the algorithm increase rapidly with the number of parts, and hence this extension of the algorithm quickly becomes impractical.

The quality of a partition generated by this algorithm depends strongly on the initial partition it is provided with. There are poor initial partitions for the regular grid graph that cannot be improved by the Kernighan-Lin algorithm [45]. Some of the other partitioning algorithms are capable of computing partitions of better quality. However, the Kernighan-Lin algorithm has found its niche in a "local" post-processing phase, to further reduce the number of edges cut, after another "global" algorithm is used to compute a good initial partition. The use of the Kernighan-Lin algorithm for this purpose has been advocated in [9, 89], and this is one of the post-processing options in CHACO [53]. Bui and Jones have implemented a multilevel Kernighan-Lin algorithm to improve the quality of the partitions computed [15, 62]. The Kernighan-Lin algorithm has been used to partition the coarsest level and to refine initial partitions within multilevel partitioning algorithms, as described in Section 6. Gilbert and Zmijewski have implemented a variant of the original Kernighan-Lin algorithm to partition a graph in parallel on distributed-memory multiprocessors [45].

3.2 Level-structure partitioning

Another early algorithm for computing vertex separators was provided in SPARSPAK [22], a library of routines for solving sparse systems of equations by direct methods. The algorithm first finds a pseudo-peripheral vertex v in the graph (one of a pair of vertices that are approximately at the greatest distance from each other in the graph). A breadth-first search from v is used to partition the vertices into levels: the vertex v belongs to the zeroth level, and all neighbors of vertices in the ith level belong to the $(i+1)$th level, for $i = 0, 1, \ldots$. The algorithm in SPARSPAK chooses the vertices in

the median level as the vertex separator. A slight variant of this algorithm chooses the separator to be the vertices in the smallest level k such that the levels $0, 1, \ldots, k$ together contain more than half the vertices. This variant partitions the vertices into roughly equal sets. Another improvement is to remove from the separator those vertices in level k that are adjacent to vertices in level $(k-1)$ but not to vertices in level $(k+1)$.

The algorithm is easily modified to compute edge separators.

This algorithm for computing vertex separators is quite fast, requiring only $\mathcal{O}(|E|)$ time, since it employs only a few breadth-first searches to compute the pseudo-peripheral vertex and the level structures. Unfortunately, the quality of the separators is quite poor, relative to the other algorithms described here [89]. Within SPARSPAK, the separator algorithm was used to compute nested dissection orderings; experience had shown that the orderings generated from this separator algorithm were not competitive with the Multiple-Minimum-Degree (MMD) orderings in terms of the storage and the arithmetic required for factoring a sparse matrix [90].

The greedy algorithm

A variant of this algorithm to partition a graph into k parts (with a requested number of vertices in the parts) has also been considered by several authors [23, 33]. As before, a pseudo-peripheral vertex is computed. The first part initially includes the pseudo-peripheral vertex, and then we include vertices in as many levels as necessary until the first part contains as many vertices as specified. We can continue this process beginning with a vertex on the boundary of the first region (such a vertex belongs either to the last level from which vertices have been included in the first part, or the next higher level if all vertices in that level have been included in the first part) to obtain the second part. The other parts can be computed by the same procedure. If some of the parts are disconnected, then the connected components in a part may be reassigned to other parts to obtain a single connected component in each part. In this case it would not be possible to achieve the requested part sizes in each part.

It is not necessary in this algorithm to compute the level structure beforehand, since a breadth-first search from the pseudo-peripheral vertex can be used to generate levels from each starting vertex.

3.3 Inertial algorithm

The inertial algorithm employs the geometrical coordinates of the vertices of a graph embedded in two or three dimensions to compute a partition. We view the set of vertices of the mesh as a discrete point set, and compute the center of gravity of this set (x_c, y_c, z_c). Then compute the 3×3 inertia matrix

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix},$$

where

$$\begin{aligned} I_{xx} &= \sum_i (y_i - y_c)^2 + (z_i - z_c)^2, \\ I_{yy} &= \sum_i (x_i - x_c)^2 + (z_i - z_c)^2, \\ I_{zz} &= \sum_i (x_i - x_c)^2 + (y_i - y_c)^2, \\ I_{xy} &= I_{yx} = -\sum_i (x_i - x_c)(y_i - y_c) \\ I_{yz} &= I_{zy} = -\sum_i (y_i - y_c)(z_i - z_c), \\ I_{xz} &= I_{zx} = -\sum_i (x_i - x_c)(z_i - z_c). \end{aligned}$$

The eigenvector $\underline{v}_1$ associated with the smallest eigenvalue of I represents the axis of minimum angular momentum. We now compute the orthogonal projection of the coordinates of the vertices onto this eigenvector $\underline{v}_1$. The median value of these projections can be used to partition the vertices into two sets. This algorithm is described in [82].

An extension to the above algorithm is to use the projections of a point on to the second and third eigenvectors of I as secondary and tertiary keys to break ties [84]. Here we form the eigenvector matrix $V = \begin{pmatrix} \underline{v}_1 & \underline{v}_2 & \underline{v}_3 \end{pmatrix}$, and then compute the product $\underline{x}^T V$, where $\underline{x}$ is the coordinate vector of a vertex. The three components of this product, in order, form the keys for partitioning.

The intuition behind this algorithm is that the rotational angular momentum is minimum about the principal axis of inertia. If the

domain is nearly convex, then this axis will align itself with the overall shape of the grid, and the grid will have a small spatial extent in a direction orthogonal to this axis. The virtue of the inertial algorithm is that it is fast, though the quality of the partitions may be poor. The quality can be improved by the use of a Kernighan-Lin post-processing phase. The algorithm also requires a geometric embedding of the graph.

4 The spectral partitioning algorithm

To motivate this algorithm, we consider the bisection problem: Partition the vertices of a graph $G = (V, E)$ into two sets A and B to minimize the number of "cut edges", i.e., edges with one endpoint in A and the other in B. The part sizes $|A| = |B| = n/2$, and the graph has an even number of vertices.

The bisection problem can be formulated as the minimization of a quadratic objective function by means of the Laplacian matrix $Q = Q(G)$ of the graph G. Let $d(i)$ denote the degree of a vertex i, i.e., the number of vertices adjacent to i. The Laplacian matrix Q has elements

$$q_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } (i,j) \in E, \\ 0 & \text{if } i \neq j \text{ and } (i,j) \notin E, \\ d(i) & \text{if } i = j. \end{cases}$$

It is easy to verify that the Laplacian Q is symmetric and has row and column sums equal to zero. It may be also expressed in terms of two other matrices associated with a graph as $Q = D - A$, where A is the adjacency matrix of a graph, and D is the $n \times n$ diagonal matrix of the degrees of the vertices of G.

Let $\underline{x}$ be an n-vector with component $x_i = 1$ if $i \in A$ and $x_i = -1$ if $i \in B$. Then

$$\sum_{(i,j)\in E} (x_i - x_j)^2 = \sum_{\substack{i\in A, j\in B, \\ (i,j)\in E}} (x_i - x_j)^2 = 4|\delta(A,B)|.$$

On the other hand,

$$\underline{x}^T Q \underline{x} = \underline{x}^T D \underline{x} - \underline{x}^T A \underline{x} = \sum_{i=1}^{n} d_i x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j = \sum_{(i,j)\in E} (x_i - x_j)^2.$$

Thus the bisection problem is equivalent to the problem of minimizing the quadratic form $\underline{x}^T Q \underline{x}$ over n-vectors $\underline{x}$ with components $x_i = \pm 1$ and $\sum_{i=1}^{n} x_i = 0$. Formally,

$$|\delta_{min}(A,B)| = \min_{\substack{x_i=\pm 1, \\ \sum_{i=1}^{n} x_i = 0}} \underline{x}^T Q \underline{x}.$$

Since the bisection problem is NP-complete, we cannot expect to solve this problem exactly. However, we can approximate this intractable problem by a tractable one if we relax the constraint that $x_i = \pm 1$ and let each component x_i vary continuously in value between $+\sqrt{n}$ and $-\sqrt{n}$. Thus we obtain the relaxed problem and its solution given below:

$$\begin{aligned} \min_{\substack{x_i=\pm 1, \\ \sum_{i=1}^{n} x_i = 0}} \underline{x}^T Q \underline{x} \quad &\geq \quad \min_{\substack{\sum_{i=1}^{n} x_i^2 = n, \\ \sum_{i=1}^{n} x_i = 0}} \underline{x}^T Q \underline{x} \\ &= \quad \underline{x_2}^T Q \underline{x_2} \\ &= \quad \lambda_2(Q) \underline{x_2}^T \underline{x_2} \\ &= \quad n\lambda_2(Q), \end{aligned}$$

where $\underline{x_2}$ is the eigenvector corresponding to the smallest positive eigenvalue of the Laplacian matrix $Q(G)$.

The minimizer of the relaxed problem is the second eigenvector of the Laplacian. We will show in Section 7 that the closest partition vector to the second eigenvector is obtained by rounding the most positive $n/2$ components of the latter to $+1$, and the remaining components to -1. Such results have also been obtained by Chan, Ciarlet, and Szeto [17].

The discussion above suggests the following algorithm to partition a graph: compute a second eigenvector of the Laplacian of the graph, and then partition the vertices into two subsets by the median eigenvector component. It should be noted that if a partition into subsets of size k and $n-k$ are desired, then the kth most positive component (or kth most negative component) could be used to obtain partition. This bipartition algorithm can be justified by the results in Section 7.

One choice of k is to put all the vertices with nonnegative eigenvector components into into one set, and all the vertices with negative components into the other set. In this case, we have the following result due to Fiedler [37].

> Let P denote the vertices of a graph G whose second eigenvector components are at least r, where $r \leq 0$. The subgraph of G induced by the set P is a connected subgraph of G. Similarly, if N denotes the vertices whose second eigenvector components are at most s, where $s \geq 0$, then the subgraph induced by N is connected.

If we have the freedom to choose the sizes of the subsets, then a good choice would be the partition that has a small isoperimetric ratio $|\delta(A,B)|/|A|$, where A is the smaller of the two subsets.

Donath and Hoffman [29] were the earliest to use spectral methods to partition graphs in the context of circuit layout. Spectral partitioning algorithms were considered and analyzed by Barnes [9], Boppana [12], Alon, Galil, and Milman [2], Mohar [80], and many others. Aspvall and Gilbert [6] used eigenvectors of the adjacency matrix to color the vertices of a graph.

Pothen, Simon, and Liou [89] used the spectral partitioning algorithm to compute separators for parallel computing, proved additional lower bounds on separators, and described computational results. Hendrickson and Leland [56] extended the spectral algorithm to compute quadrisections and octasections.

Work on eigenvalues and eigenvectors of graphs can be found in the books [25, 26], and in the surveys by Mohar and Poljak [80, 81].

5 The geometric algorithm

Finite element or finite difference meshes embedded in space contain geometric information about the coordinates of the mesh points. Algorithms for partitioning meshes by bisecting along coordinate axes have been considered by Simon [96], Williams [105], and many others. A parallel nested dissection algorithm based on this idea has been described by Heath and Raghavan [52]. These algorithms have the virtue of being fast, and are easy to implement in parallel; however, the quality of the separators obtained by such straight-line cuts are not good relative to the other algorithms, especially for adapted meshes.

We now describe a geometric partitioning algorithm designed by Miller, Teng, Thurston, and Vavasis [79], and implemented by Gilbert, Miller, and Teng [44]. This algorithm computes a separator by using a circle rather than a straight-line to cut the mesh. Given

a graph embedded in d-dimensional space we disregard the edges of the graph, and view the graph as a collection of vertices. Since the graph is embedded in d-dimensional space, each vertex has a set of geometric coordinates attached to it.

A *centerpoint* of a given set of points is a point such that every hyperplane through it divides the given set of points approximately evenly into two subsets. "Approximately evenly" in this case means that the worst-case ratio of the sizes of the two subsets is $d : 1$. It can be proved that every finite point set in $\Re^d$ has a centerpoint, and the proof yields a polynomial time algorithm that employs linear programming to compute the centerpoint. However, this algorithm is too slow to be practical, and heuristics to compute approximate centerpoints are used instead.

For convenience in the following exposition we first scale and translate the coordinates of the given points so that the transformed coordinates are between -1 and $+1$. An outline of the geometric partitioning algorithm is as follows:

1. Project the input points in $\Re^d$ to the surface of the unit sphere in $\Re^{d+1}$ centered at the origin. The "north pole" of the sphere has coordinates $(0, 0, \ldots, 0, 1)$, and a point p is projected to the surface of the sphere along the line through p and the north pole.

2. Compute a centerpoint of the projected points on the surface of the $d+1$-dimensional sphere. The centerpoint is in the interior of the sphere.

3. Move the centerpoint to the origin of the sphere in two steps: First, rotate the projected points about the origin in $\Re^{d+1}$ to make the centerpoint a point $(0, 0, \ldots, 0, r)$ on the $(d + 1)$-st axis. Second, dilate the points on the surface of the sphere to make the centerpoint the origin.

4. Choose a random great circle on the unit sphere in $\Re^{d+1}$.

5. Transform the great circle in $\Re^{d+1}$ to a a circle in $\Re^d$ by reversing the dilation, rotation, and projecting back from $\Re^{d+1}$ to $\Re^d$.

6. A vertex separator is the set of vertices that lie sufficiently "close" to the separating circle. ("Close" means that the circle

intersects the balls corresponding to the vertices, when the balls are expanded by a factor α. See the description of overlap graphs in Section 2.) An edge separator is the set of edges cut by the separating circle.

The geometric separator algorithm guarantees the size of the vertex separator S in an overlap graph (see the definition in Section 2) embedded in d-dimensions is $\mathcal{O}(n^{(d-1)/d})$ with high probability. If S separates the graph G into two parts A and B, then the larger of the two parts has at most $(d+1)/(d+2)$ vertices. These results hold for finite element meshes in two or three dimensions when the aspect ratios of the elements are bounded. It is too expensive to implement an algorithm providing this guarantee, and a practical implementation makes use of several heuristic ideas for faster computation of the separators. The practical implementation does not yield a separator guaranteed to be small; nevertheless, computational results show that the separators computed by the practical algorithm are comparable in quality relative to other algorithms.

We now discuss the heuristic ideas used by Gilbert et al. in their practical implementation of the geometric algorithm [44]. We begin by describing how a centerpoint is approximated.

The approximate centerpoint computation is expensive, and hence it is important to reduce the sample size to obtain a fast partitioning algorithm. Thus a randomly chosen sample of the vertices, viewed as points in d-dimensional space, is used to compute the approximate centerpoint. (Gilbert et al. suggest $(d+3)^4$ points suffice in d dimensions; this works out to 625 points in two dimensions and 1296 points in three dimensions.) The sample size reduction sacrifices the guaranteed worst-case splitting ratio of $d+1:1$, although empirically the centerpoint approximation is a good one.

The approximate centerpoint is computed by repeatedly finding Radon points of $d+3$ points. A point q is a Radon point of a set P of points if there is a partition of P into sets P_1 and P_2 such that q belongs to the intersection of the convex hull of P_1 and the convex hull of P_2.

Every set of $d+3$ points in $\Re^{d+1}$ has a Radon point, which can be computed in the following manner. Let $P = \{p_1, \ldots, p_{d+3}\}$, with the coordinates of p_i given by p_i^j, for $j = 1, \ldots, d+1$. Consider the

set of $d+2$ homogeneous equations in the $d+3$ unknowns α_i

$$\sum_{i=1}^{d+3} \alpha_i p_i^j = 0, \quad \text{for } j = 1, \ldots, d+1; \quad \sum_{i=1}^{d+3} \alpha_i = 0.$$

Since this system is underdetermined, there is always a nonzero solution; further, by the last equation, some of the α_i are positive, and some negative. Thus the sets $P_1 = \{i : \alpha_i > 0\}$ and $P_2 = \{i : \alpha_i \leq 0\}$ are both nonempty. Let $c = \sum_{i \in P_1} \alpha_i$. Then the point q can be expressed as a convex combination of points in P_1 and P_2.

$$q = \sum_{i \in P_1} (\alpha_i / c) p_i \quad = \quad \sum_{i \in P_2} (-\alpha_i / c) p_i.$$

The point q can be computed in this manner from a null vector of the coefficient matrix of the homogeneous system of equations.

The approximate centerpoint is computed by repeatedly placing a random sample of the input points into a queue; groups of $d+3$ points from the head of the queue are replaced by their Radon points, which are inserted into the tail of the queue. Repetition of this process leaves a single point in the queue, which is then accepted as an approximate center point.

We now turn to the choice of a random great circle of the sphere in $\Re^{d+1}$ to partition the point set. Since a great circle may not yield a partition into roughly equal size sets, a random circle is used to obtain an even split. A circle of the sphere in $\Re^{d+1}$ is represented by its normal vector. Since the circle is chosen randomly, this means that we choose a random vector. If v is a random normal vector, and $p_1, \ldots, p_n$ are the points (all viewed as row vectors), then an even split is obtained by partitioning with respect to the median value of the inner products vp_i^T. This corresponds geometrically to shifting the separating plane away from the origin, in a direction normal to the plane, until it evenly splits the mapped points in the sphere. Since the separating circle is a circle and not a great circle on the unit sphere, the theoretically expected separator size may not hold. In practice the mesh is separated by generating a number of random separating circles, and the best partition obtained is recorded. (Gilbert et al. report that about thirty trials suffice usually to obtain a good separator.)

In the practical algorithm the edge separator is the set of edges cut by the separating circle. A vertex separator is computed by

finding a minimum vertex cover of the bipartite graph induced by the edge separator.

The geometric algorithm has several advantages. It examines only the vertices of the graph, and makes no use of the edges except to compute the quality of the generated separators. The computations involved (projecting up to $\Re^{d+1}$, finding an approximate centerpoint, rotation and dilation of the points, projecting down) involve simple operations on the points, and the repeated computation of the null vector of a $(d+2)$ by $(d+3)$ matrix. This also makes the algorithm attractive on a parallel computer. However, the feature that the algorithm makes no use of the edge information in the graph is also a weakness in the partitioning of edge-weighted meshes where the weight of the cut edges needs to be minimized. Another disadvantage is that graphs arising in many areas, such as econometric modeling, do not have coordinate information since they are not embedded in space.

One can apply the geometric algorithm to problems that do not have coordinate information if it is possible to embed the problem in space. Hall [49] has shown that the eigenvectors of the Laplacian matrix associated with a graph can be used to embed the graph. Chan, Gilbert, and Teng [18] have reported results on partitioning graphs by the geometric algorithm, with coordinates generated from the Laplacian eigenvectors.

6 A multilevel algorithm

Multilevel algorithms for graph partitioning are similar in spirit to multigrid algorithms for solving linear systems of equations. We begin by providing an overview of the algorithm: details are spelled out as the individual steps are discussed. We view the given graph, with a large number of vertices, as the finest graph in a sequence of coarse graphs to be computed. Given a "fine" graph, we obtain a "coarse" graph with fewer vertices by a suitable shrinking procedure. We construct a sequence of "coarse" graphs until the coarsest graph computed thus far is small enough. A high-quality partitioning algorithm such as the spectral algorithm or Kernighan-Lin algorithm is used to partition the coarsest graph. A partition of a coarse graph is then used to partition the fine graph immediately preceding it in the sequence of graphs by reversing the shrinking step used to coarsen

(this is an "uncoarsening" step). Next, this "rough" partition of the fine graph is refined by means of a vertex-swapping algorithm that moves vertices between the parts to reduce the number of edges cut by the partitioning algorithm (this is a "refinement" step). The uncoarsening and refinement steps are repeated for each successive pair of fine and coarse graphs in the sequence until a partition of the given graph is computed.

Some of the ideas involved in the multilevel algorithm were used to obtain efficient implementations of the Kernighan-Lin algorithm by Bui et al. [15], who called it graph compaction. Barnard and Simon [8] described a spectral partitioning algorithm in which they used a multilevel algorithm to compute the Fiedler vector of a given graph, by coarsening, interpolating, and then refining. This algorithm was further improved by Van Driessche and Roose [99]. John G. Lewis (personal communication) observed that a partition of a fine graph can be computed directly from a partition of the coarse graph by projecting the separator of the coarse graph to the fine graph and then refining it. Hendrickson and Leland [55] implemented such an algorithm, and included it in their CHACO software [53]. We follow them in the detailed description of the steps a multilevel partitioning algorithm.

The multilevel algorithm has also been implemented by Karypis and Kumar [64]; they have further refined the algorithm, and included an implementation in their METIS software. They have also described a parallel implementation of this algorithm and provided an analysis [63, 65]. A multilevel partitioning algorithm has been implemented by Kumfert and Pothen [69], and results from a nested dissection ordering computed with a multilevel algorithm are included in Section 8.

Coarsening step. Barnard and Simon coarsened a graph by finding a subset of non-adjacent vertices S, and then "growing" neighborhoods around each vertex in S. The coarse graph has the vertex set equal to S, and an edge joins two vertices in this graph when the corresponding neighborhoods intersect. (The set of nodes S can be chosen to be a maximal independent set of vertices in the fine graph. An *independent set* is a set of pairwise non-adjacent vertices. An independent set is *maximal* if no other vertex can be included in the set while preserving the property of non-adjacency.)

Another way to coarsen a graph is by contracting the two endpoints of an edge and replacing them by a single vertex in the coarse

graph. The new vertex is adjacent to all neighbors of the two vertices it replaces. What about a common neighbor of the two merged vertices? The information that this neighbor is adjacent to both the merged vertices, and not just to one of them, can be used to compute a partition of the coarse graph that will lead to a better partition of the uncoarsened graph. One way of capturing this information is by assigning weights to the vertices and edges of the graph. (An initial unweighted graph is equivalent to a graph with all vertex and edge weights equal to one.) When the two endpoints of a contracted edge have a common neighbor, the new edge joining the neighbor to the new vertex has a weight equal to the sum of the weights of the two original edges replaced by the new edge. When a vertex is a neighbor of only one of the two merged vertices, an edge joins that neighbor to the new merged vertex, and its weight is unchanged. The weight of the new vertex obtained by contracting an edge is the sum of the weights of the two endpoints.

One choice of a set of edges to contract is a matching in the graph. A *matching* is a subset of edges in a graph such that at most one edge in the subset is incident on each vertex in the graph. Thus no two edges in a matching can share a common endpoint. A *maximal matching* is a matching whose cardinality cannot be increased by adding another edge to it while preserving a matching. A good choice for the set of edges to be contracted is a maximal matching. Hendrickson and Leland compute a maximal matching by matching the vertices in a random order to make the sequence of coarse graphs constructed insensitive to the particular maximal matching computed. Karypis and Kumar [64] prefer to match a vertex to an unmatched neighbor joined to it by a heaviest edge, and consider vertices to match in a random order.

Uncoarsening step. During the coarsening step we record the vertex weights and the edge weights of each coarse graph. A vertex in a coarse graph corresponds to a unique set of merged vertices in the fine graph, and hence it is possible to compute a partition of the latter from a partition of the former.

The coarsening procedure by means of matchings described here has two important properties:

1. The total weight of the edges cut by a partition in the coarse graph is equal to the total weight of the edges cut in the fine graph when that partition is projected from the former to the

latter.

2. The sum of the node weights is unchanged in the fine and coarse graphs.

We maintain load balance by ensuring that the sum of the node weights in each part are roughly equal; we maintain low communication costs by keeping the sum of the weights of the cut edges low. The invariance of the total vertex weights in each part and the sum of the weights of the cut edges under the coarsening and uncoarsening steps ensure that a good partition of the coarse graph is also a reasonable initial partition of the fine graph.

Partitioning the coarsest graph. The coarsening process stops when the coarsest graph typically has a few hundred vertices, or the coarsening process does not reduce the number of vertices significantly. The coarsest graph can be partitioned by one of the many partitioning algorithms discussed in this section. The spectral partitioning algorithm suitable for a vertex- and edge-weighted graph could be used, and this is one of the options in CHACO. The Kernighan-Lin algorithm is another possibility, and since the graph is quite small, many more nodes could be considered for moves. An initial partition could be obtained in this case by a level-structure based partitioner that has been modified to partition weighted graphs.

Refinement step. The Kernighan-Lin algorithm is used to refine the partition projected to a fine graph from a partition of the coarse graph. This is an ideal situation for applying the Kernighan-Lin algorithm because the partition projected from the coarse graph is a reasonable initial partition of the fine graph, and thus the algorithm should be able to obtain a local minimum in the partition space relatively quickly.

A few observations can be used to speed up this part of the computation. One is that gains need to be computed only for the "boundary vertices", i.e., those vertices incident on the edges cut by the partition. As vertices are moved, non-boundary vertices that are neighbors of boundary vertices may themselves become new boundary vertices; the gains of such vertices need to be computed only when they become boundary vertices. Another observation is that when a preset number of vertex moves fails to improve the number of cut edges, then the algorithm can be terminated, if we assume that the refinement procedure begins with a reasonably good initial partition.

Current experience suggests that the multilevel algorithm computes partitions of good quality, and is fast, both relative to the other partitioning algorithms.

7 Bipartition and the quadratic assignment problem

In this section we justify the spectral partitioning algorithm by a mathematical programming formulation of the bipartitioning problem. This work is due to Rendl and Wolkowicz [92] who described their results in terms of the adjacency matrix perturbed by a diagonal matrix. This exposition in terms of the Laplacian matrix is from [86]. Readers who are not enamored by theoretical issues could skip this section and go on to the next section on applications of partitioning.

Consider the problem of partitioning a graph $G = (V, E)$ into two subgraphs of m_1 and m_2 vertices such that the number of edges joining one subgraph to the other, is minimized. We denote the two vertex subsets by V_1 and V_2, with $|V_j| = m_j$ for $j = 1, 2$, and $m_1 + m_2 = n$. This problem can be formulated as a quadratic assignment problem (QAP) involving the Laplacian matrix Q of the graph G. Recall that the Laplacian $Q = D - A$, where D is a diagonal matrix of vertex-degrees, and A is the adjacency matrix of G.

7.1 Formulation of bipartition as a QAP

Let $X = \begin{pmatrix} \underline{x}_1 & \underline{x}_2 \end{pmatrix}$ be an $n \times 2$ *partition matrix* consisting of the two indicator vectors $\underline{x}_j$ (for $j = 1, 2$), where x_{ij} is equal to one if vertex i belongs to the set V_j, and is zero otherwise. Then

$$\underline{x}_j{}^T Q \underline{x}_j = \sum_{i=1}^{n} \sum_{k=1}^{n} x_{ij} q_{ik} x_{kj} = \sum_{v \in V_j} d(v) - 2|E(V_j, V_j)|,$$

where $d(v)$ is the number of vertices adjacent to v, and $E(V_j, V_j)$ is the set of edges in E with both endpoints in V_j. We denote the edges joining V_1 and V_2 by the set $\delta(V_1, V_2)$, and recall that the *trace* of a square matrix Q, denoted by $tr\ Q$, is the sum of its diagonal elements. Then

$$tr\ X^T Q X \quad = \quad \sum_{v \in V} d(v) - 2|E(V_1, V_1)| - 2|E(V_2, V_2)|$$

$$\begin{aligned} &= 2|E| - 2|E(V_1, V_1)| - 2|E(V_2, V_2)| \\ &= 2|\delta(V_1, V_2)|. \end{aligned} \tag{1}$$

Thus the problem of minimizing the number of edges cut by a bipartition with part sizes equal to m_1 and m_2 can be written as

$$\begin{aligned} |\delta_{min}(V_1, V_2)| &\equiv \min\{|\delta(V_1, V_2)| : |V_1| = m_1, |V_2| = m_2\} \\ &= (1/2) \min_X tr\ X^T Q X, \end{aligned} \tag{2}$$

where X varies over partition matrices with exactly m_j ones in the jth column.

Let $\underline{u}_n = (1/\sqrt{n}) \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^T$ denote the n-vector of all ones, scaled to have unit 2-norm. (We will write $\underline{u}$ instead of $\underline{u}_n$ when the dimension is clear from the context.) A partition matrix X is characterized by the following three conditions:

$$X\underline{u}_2 = \sqrt{(n/2)}\,\underline{u}_n; \quad X^T\underline{u}_n = (1/\sqrt{n}) \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}; \tag{3}$$

$$X^T X = \begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix} \equiv M; \tag{4}$$

$$x_{ij} \geq 0 \quad \text{for } i = 1, \ldots, n; j = 1, 2. \tag{5}$$

The first part of the first condition states that each row sum of a partition matrix is one, signifying that each vertex belongs to exactly one of the parts V_1 or V_2. The second part shows that there are m_j vertices in the jth part V_j. The second condition indicates that the columns of a partition matrix are orthogonal, and the third that the elements of a partition matrix are nonnegative.

Scaling $X = YM^{1/2}$ simplifies the following exposition and exposes the structure of the problem. With this scaling, the conditions on X are transformed to the following conditions on Y:

$$\begin{aligned} Y\underline{m} &= \underline{u}_n; \quad Y^T\underline{u}_n = \underline{m}, \\ &\quad \text{where} \quad \underline{m} = (1/\sqrt{n}) \begin{pmatrix} \sqrt{m_1} \\ \sqrt{m_2} \end{pmatrix}; \end{aligned} \tag{6}$$

$$Y^T Y = I_2; \tag{7}$$

$$\left(YM^{1/2}\right)_{ij} \geq 0 \quad \text{for } i = 1, \ldots, n;\ j = 1, 2. \tag{8}$$

The objective function $tr\ X^TQX$ becomes $tr\ M^{1/2}Y^TQY\,M^{1/2} = tr\ MY^TQY$. In the last transformation we have used the identity $tr\ MN = tr\ NM$, where M is $n \times k$, and N is $k \times n$.

Minimizing this objective function subject to these constraints is NP-complete. Hence we obtain lower bounds on the number of edges cut by relaxing the third of these conditions.

7.2 Projected lower bounds

It is convenient to impose (6) on Y by projecting the problem to the subspace orthogonal to the manifold defined by this condition. Note that the two parts of this condition yield $YY^T\underline{u}_n = \underline{u}_n$, and $Y^TY\underline{m} = \underline{m}$. Thus we find that $\underline{m}$ is a right singular vector and that $\underline{u}_n$ is a left singular vector of Y corresponding to the singular value one. Choose an $n \times n$ orthogonal matrix $P_1 = \begin{pmatrix} \underline{u} & V \end{pmatrix}$, and a 2×2 orthogonal matrix $P_2 = \begin{pmatrix} \underline{m} & \underline{v} \end{pmatrix}$. The first step of the singular value decomposition of Y is

$$\begin{aligned} P_1^TYP_2 &= \begin{pmatrix} \underline{u}^TY\underline{m} & \underline{u}^TY\underline{v} \\ V^TY\underline{m} & V^TY\underline{v} \end{pmatrix} = \begin{pmatrix} \underline{u}^T\underline{u} & \underline{m}^T\underline{v} \\ V^T\underline{u} & V^TY\underline{v} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ \underline{0} & \underline{z} \end{pmatrix}, \quad \text{where } \underline{z} \equiv V^TY\underline{v}. \end{aligned} \tag{9}$$

Thus if we choose Y to be

$$Y = P_1\begin{pmatrix} 1 & 0 \\ \underline{0} & \underline{z} \end{pmatrix}P_2^T = \underline{u}\,\underline{m}^T + V\underline{z}\,\underline{v}^T, \tag{10}$$

then (6) is satisfied.

Substitution of this representation of Y in the orthogonality condition (7), followed by pre-multiplication with P_2^T and post-multiplication with P_2, shows that

$$\begin{pmatrix} 1 & 0 \\ 0 & \underline{z}^T\,\underline{z} \end{pmatrix} = I_2,$$

and hence we obtain the condition $\underline{z}^T\,\underline{z} = 1$.

Substituting for Y from (10) in the objective function $trMY^TQY$, we find that since $Q\underline{u} = \underline{0}$, only one of the four terms survives, and it becomes

$$\begin{aligned} tr\ MY^TQY &= tr\ M\underline{v}\,\underline{z}^TV^TQV\underline{z}\,\underline{v}^T = tr\ \underline{v}^TM\underline{v}\,\underline{z}^T\widehat{Q}\underline{z} \\ &= (\underline{v}^TM\underline{v})\,(\underline{z}^T\widehat{Q}\underline{z}), \end{aligned} \tag{11}$$

where $\widehat{Q} \equiv V^T Q V$ is the projected Laplacian.

The first term on the right-hand-side, $\underline{v}^T M \underline{v}$, is easily computed to be $2m_1m_2/n$, since $\underline{m} = (1/\sqrt{n})\begin{pmatrix} \sqrt{m_1} \\ \sqrt{m_2} \end{pmatrix}$ implies that $\underline{v} = \pm(1/\sqrt{n})\begin{pmatrix} \sqrt{m_2} \\ -\sqrt{m_1} \end{pmatrix}$. Thus we obtain the result

$$|\delta(V_1, V_2)| = (1/2)\,(2m_1m_2/n)\,\underline{z}^T \widehat{Q} \underline{z}.$$

The bipartition problem is hence

$$\begin{aligned} |\delta_{min}(V_1, V_2)| &= (m_1m_2/n) \min_{\underline{z}} \underline{z}^T \widehat{Q} \underline{z} \\ \text{subject to} \quad & \underline{z}^T \underline{z} = 1, \\ & \left(\underline{u}\,\underline{m}^T M^{1/2} + V \underline{z}\,\underline{v}^T M^{1/2}\right)_{ij} \geq 0. \end{aligned} \tag{12}$$

Though this problem is intractable, a lower bound may be obtained by relaxing the second constraint, and thus we find

$$|\delta_{min}(V_1, V_2)| \geq (m_1m_2/n)\lambda_1(\widehat{Q}) = (m_1m_2/n)\lambda_2(Q), \tag{13}$$

since the eigenvalues of $\widehat{Q}$ are the $n-1$ nonzero eigenvalues of Q. The lower bound is attained by the corresponding eigenvector $\underline{z}_0 = V^T \underline{x}_2$, where $\underline{x}_2$ is the second Laplacian eigenvector. Hence the orthogonal matrix attaining the lower bound is

$$Y_0 = \underline{u}\,\underline{m}^T + V V^T \underline{x}_2\,\underline{v}^T. \tag{14}$$

7.3 Diagonal perturbations

The lower bound on the number of cut edges can be improved further by considering diagonal perturbations of the Laplacian. Let $Q(\underline{d}) = Q + Diag(\underline{d})$, where $\underline{d}$ is an n-vector whose components sum to zero. It can be verified that $tr\, X^T Q(\underline{d}) X = tr\, X^T Q X$, so that this perturbation has no effect on the number of cut edges. Proceeding as in the unperturbed case, we can show that

$$\begin{aligned} |\delta_{min}(V_1, V_2)| \;\geq\; & \max_{\underline{d}} \min_{\underline{z}} \Big\{ (m_1m_2/n) \underline{z}^T \widehat{Q}(d) \underline{z} \\ & + \; (1/2n\sqrt{n})(m_1 - m_2)\sqrt{m_1m_2}\,\underline{d}^T\,\underline{z} \Big\}, \\ \text{subject to} \quad & \underline{z}^T \underline{z} = 1. \end{aligned} \tag{15}$$

The lower bound can be computed by nondifferentiable optimization techniques [94].

The lower bounds in terms of the unperturbed Laplacian and the perturbed adjacency matrix, described by Falkner et al. [32], are still weak for finite-element meshes with bounded aspect ratios.

7.4 Closest partition matrix

Which partition matrix Z is "closest" to the orthogonal matrix $X_0 = Y_0 M^{1/2}$ attaining the lower bound (see (14)) in the bipartition problem? We can answer this question by considering the objective function of the bipartition problem, which we may write as

$$\min_Z \; tr\; Z^T(Q+\alpha I)Z = \min_Z \; \|(Q+\alpha I)^{1/2}Z\|_F{}^2. \qquad (16)$$

Here we have shifted the Laplacian by a small positive multiple of the identity to make the matrix $Q+\alpha I$ positive definite, so that its square root is nonsingular. This is necessary to obtain a weighted norm. It can be verified that this shifts the objective function by the constant $\alpha\, n$, and hence has no effect on the minimizer. We now expand Z about X_0 to obtain

$$\begin{aligned} &\min_Z \; \|(Q+\alpha I)^{1/2}(X_0+(Z-X_0))\|_F{}^2 \\ &= \min_Z \; \|(Q+\alpha I)^{1/2}X_0\|_F{}^2 + 2\, tr\; X_0^T(Q+\alpha I)(Z-X_0) \qquad (17) \\ &+ \|(Q+\alpha I)^{1/2}(Z-X_0)\|_F{}^2. \end{aligned}$$

Here we have used the identity

$$\|A+B\|_F{}^2 = \|A\|_F{}^2 + \|B\|_F{}^2 + 2\, tr\; A^T B,$$

for real matrices A and B.

On the right-hand-side of (17), the first term is a constant since X_0 is a fixed orthogonal matrix; we ignore the third term, which is a quadratic in the difference $(Z-X_0)$, to obtain a linear approximation. Hence we consider the problem

$$\min_Z \; tr\; X_0^T(Q+\alpha I)Z \quad = \quad \min_Z \; tr\; M^{1/2}Y_0^T(Q+\alpha I)Z. \qquad (18)$$

Substituting for Y_0 from (14), and noting that $\underline{u}^T Q = \underline{0}^T$, the problem becomes

$$\min_Z \; tr\; M^{1/2}\left(\underline{v}\,\underline{x}_2{}^T VV^T QZ + \alpha\,\underline{m}\,\underline{u}^T Z + \alpha\,\underline{v}\,\underline{x}_2{}^T VV^T Z\right). \qquad (19)$$

The second term in the right-hand-side is constant since $\underline{u}^T Z = (1/\sqrt{n}) \begin{pmatrix} m_1 & m_2 \end{pmatrix}$ (by (3)), and hence the problem reduces to

$$\min_Z tr\ M^{1/2} \underline{v}\, \underline{x}_2{}^T V V^T (Q + \alpha I) Z. \tag{20}$$

Replacing $VV^T = P_1 P_1^T - \underline{u}\,\underline{u}^T = (I_n - \underline{u}\,\underline{u}^T)$, noting again that $\underline{u}$ is an eigenvector of Q corresponding to the zero eigenvalue (or that $\underline{x}_2{}^T \underline{u} = 0$), the objective function simplifies to

$$\min_Z\ tr\ M^{1/2} \underline{v}\, \underline{x}_2{}^T (Q + \alpha I) Z = \min_Z\ (\lambda_2(Q) + \alpha)\, tr\ M^{1/2} \underline{v}\, \underline{x}_2{}^T Z.$$

Further simplifications are possible. An important observation is that in the bipartition problem $Z = \begin{pmatrix} \underline{z}_1 & \sqrt{n}\underline{u} - \underline{z}_1 \end{pmatrix}$, where $\underline{z}_1$ is an indicator vector with m_1 ones and remaining elements equal to zeros, and the second column of Z is the complement of $\underline{z}_1$ with respect to the vector of all ones. Also note that $M^{1/2}\underline{v} = \pm\sqrt{m_1 m_2/n} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Putting these observations together, the problem becomes

$$\left((\lambda_2(Q) + \alpha)\sqrt{m_1 m_2/n}\right) \cdot$$

$$\min_{\underline{z}_1}\ tr\ \pm \begin{pmatrix} \underline{x}_2{}^T \underline{z}_1 & \underline{x}_2{}^T(\sqrt{n}\underline{u} - \underline{z}_1) \\ -\underline{x}_2{}^T \underline{z}_1 & -\underline{x}_2{}^T(\sqrt{n}\underline{u} - \underline{z}_1) \end{pmatrix} \tag{21}$$

$$= \left(2(\lambda_2(Q) + \alpha)\sqrt{m_1 m_2/n}\right) \min_{\underline{z}_1}\ \pm \underline{x}_2{}^T \underline{z}_1. \tag{22}$$

In going from equation (21) to (22) we have used $\underline{x}_2{}^T \underline{u} = 0$.

Thus the algebraic manipulations in this subsection come to a glorious conclusion! One solution to (22), and hence to a linear approximation to the nearest partition matrix problem, is obtained by choosing $\underline{z}_1$ to have ones in rows corresponding to the smallest (most negative) m_1 eigenvector components of $\underline{x}_2$. A second solution is obtained by choosing the rows corresponding to the largest (most positive) m_1 eigenvector components. (These two solutions correspond to the choice of the sign in (22).) Hence we obtain a justification of the spectral algorithm for the bipartition problem which partitions the graph with respect to the m_1th smallest or largest second eigenvector component.

8 Applications of Graph Partitioning

Graph partitioning algorithms have been used in several applications ranging from structural mechanics to VLSI design. We do not attempt a comprehensive discussion of these here, but list some of the papers that discuss applications. After doing so, we will move on to the primary application considered in this section, the computation of nested dissection orderings by the multilevel partitioning algorithm.

8.1 A list of applications

One of the earliest applications of graph partitioning that spurred the development of the Kernighan-Lin algorithm [66] and the work of Donath and Hoffman [29] was circuit layout. Graph and hypergraph partitioning algorithms for VLSI design and layout have been surveyed by Alpert and Kahng [3].

Mesh partitioning algorithms for solving discretizations of partial differential equations on parallel computers are described in [20, 105]. The parallel solution of Euler equations and other CFD problems are described in [28, 59, 95, 102]. A survey of parallel computing in CFD has been provided in [93]. Applications in structural mechanics have been considered in [58, 82, 100]. A comparative study of partitioners in domain decomposition has been provided in [24]. Dynamic load balancing for time-dependent partial differential equations has been considered in [57, 101, 103].

8.2 Spectral Nested Dissection

Nested dissection is a divide-and-conquer scheme for ordering sparse symmetric positive definite systems of equations for fast solution by direct methods. At each step of nested dissection, a vertex separator in the adjacency graph of the sparse matrix of coefficients is computed, and the vertices in the separator are numbered with the highest available numbers. (Thus the corresponding columns in the matrix are factored *after* factoring columns in the two parts.) Then the vertices in the two parts are numbered recursively by the same strategy. Nested dissection and generalizations have been described by George [40], Lipton and Tarjan [74], and Gilbert [43].

A good vertex separator in this context should satisfy two criteria: First, the size (number of vertices) of the separator should be

small. Second, the two parts obtained by removing the separator should be roughly balanced in size. The first requirement controls the work and storage needed for the factorization, since it can be shown that the vertex separator becomes a complete graph (the corresponding submatrix becomes dense during the factorization). The second requirement ensures that there is a high degree of parallelism in the parallel algorithm by controlling the height of a data structure called the elimination tree. The height of the tree counts the number of steps in a parallel factorization algorithm, and thus measures the maximum amount of parallelism available in the parallel factorization.

We had described a spectral nested dissection algorithm (SND) in earlier work [90], in which the separator was computed by a spectral partitioning algorithm. We first compute, using the spectral algorithm, an edge separator of small size that partitions the graph into parts that are nearly balanced; a matching algorithm is then used to compute a vertex separator from the given edge separator. A minimum vertex cover of the bipartite graph induced by the edge separator is a vertex separator of the smallest size that can be obtained from the given edge separator. The vertices in the separator are ordered last, and the same strategy is recursively employed to partition the two parts.

Using the SND ordering, on a 64-processor Intel Paragon, we were able to factor a collection of sparse matrices faster than when the widely used Multiple-Minimum-Degree (MMD) ordering was used [88]. On most of these problems, the spectral ordering led to less fill, and fewer arithmetic operations in the factorization; but even on the problems where the SND ordering created more work than the MMD ordering, the decrease in the height of the elimination tree (and hence the critical path in the parallel factorization) in the SND ordering compensated for the increased work, and led to faster factorization times. On the largest problem in this test set, COPTER2 (a problem with about 55 thousand nodes and 870 thousand nonzeros), the SND ordering speeded up the factorization by a factor greater than two.

Our results on SND established for the first time that nested dissection algorithms could be competitive with the MMD algorithm for sparse matrix factorization on both serial and parallel computers. Unfortunately, the earlier level-structure-based separator algorithm available in SPARSPAK had not produced results of comparable quality. Indeed, the first attempts at computing good orderings

for parallel factorization were based on improving the concurrency in an MMD ordering by computing a perfect elimination reordering of the filled graph from the MMD ordering; for instance, see [73].

The SND ordering had the drawback that it was slower than the MMD ordering by a factor of five to twelve on a vector computer such as the Cray-Y/MP. On workstations without the vector floating point hardware, SND was even slower. The Recursive Spectral Bisection (RSB) algorithm, used when edge separators are desired, suffered from the same drawback. This led to attempts to design faster nested dissection and partitioning algorithms.

8.3 Multilevel Nested Dissection

Multilevel Nested Dissection (MLND) computes the nested dissection ordering by recursively partitioning the graph using the multilevel graph partitioning algorithm.

Results from multilevel nested dissection algorithms have been obtained by Hendrickson and Rothberg (personal communication, 1994), Karypis and Kumar [64], and by Kumfert and Pothen [69]. We compare three algorithms: the Multiple-Minimum-Degree ordering of Liu [76], an MLND algorithm due to Kumfert and Pothen [69], and MLNDW, a weighted variant of the MLND algorithm, due to Karypis and Kumar, included in MeTiS version 2.0. The multilevel nested dissection algorithms differ in the way the coarse graphs are constructed. Both variants coarsen the graph by shrinking the edges in a maximal matching; the matching is obtained by matching each unmatched vertex to one of its currently unmatched neighbors. In the MLND algorithm, the unmatched neighbor is chosen randomly; in the weighted MLNDW algorithm, it is chosen to be the unmatched neighbor joined to the vertex by a heaviest edge, the choice advocated by Karypis and Kumar.

We report the number of arithmetic operations (work) in a sparse Cholesky factorization in Figure 2, and the height of the elimination tree (the number of steps in a parallel factorization) in Figure 3. Each Figure shows a histogram obtained by normalizing the results for each problem with respect to the MMD algorithm. Results for seventeen problems from structural analysis, CFD, and linear programming applications are reported. The problems range in order from about a thousand to a hundred thousand. Information about each problem is included in Table 1.

Table 1: The list of test problems.

Problem	$\|V\|$	$\|E\|$	Comment
fischer	823	7003	
barth	6,691	19,748	
barth4	6,019	17,473	CFD problems
barth5	15,606	45,878	
commanche_dual	7,920	11,880	
shuttle.eddy	10,429	46,585	
copter1	17,222	96,921	
copter2	55,476	352,238	
tandem_vtx	18,454	117,448	
tandem_dual	84,069	183,212	
ford1	18,728	41,424	Structural problems
ford2	100,196	222,246	
onera_dual	85,567	116,817	
pds10	16,558	66,550	
ken13	28,632	95,218	Linear programs
finance256	37,376	130,560	

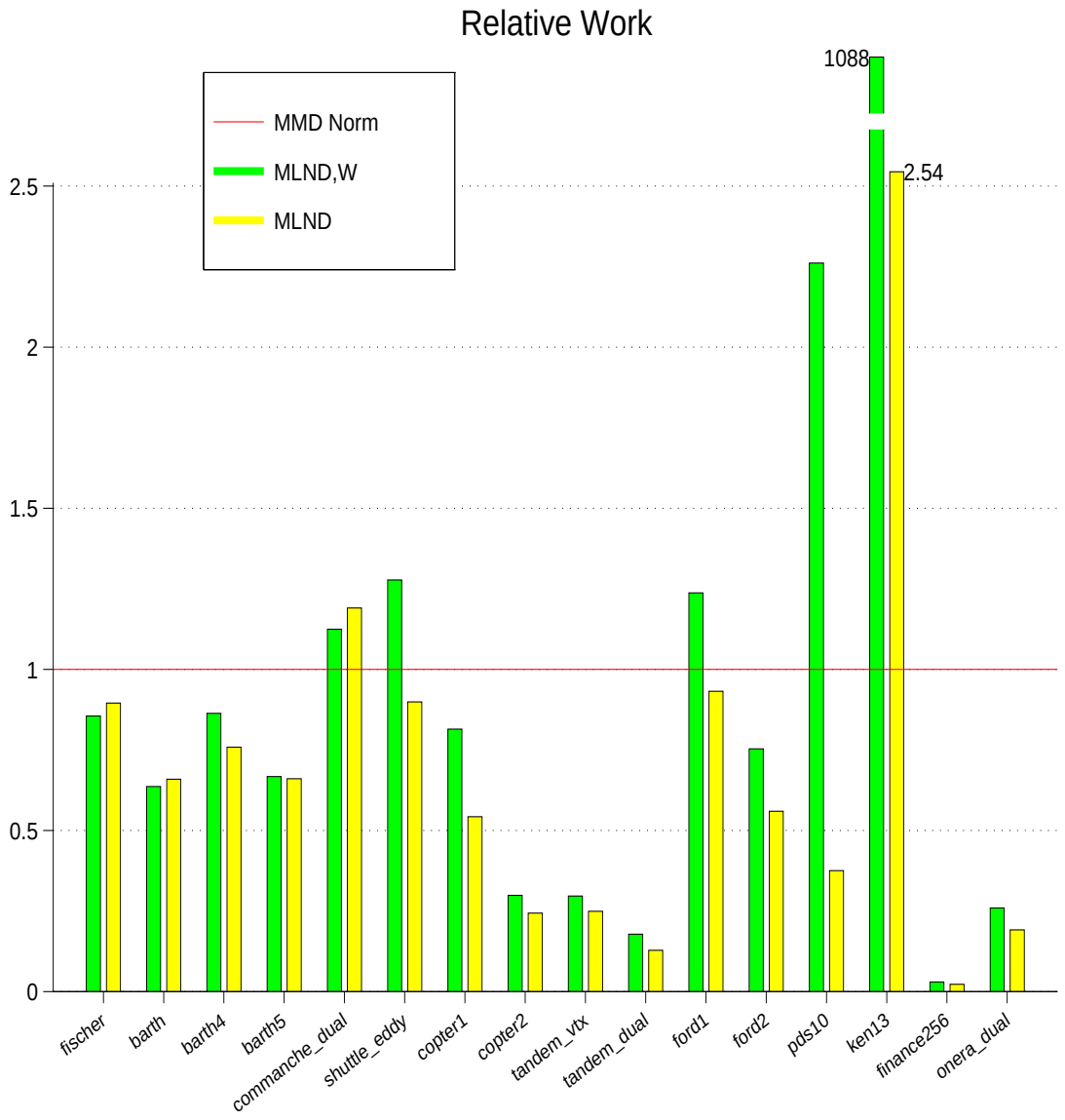

Figure 2: Work of multilevel ordering algorithms.

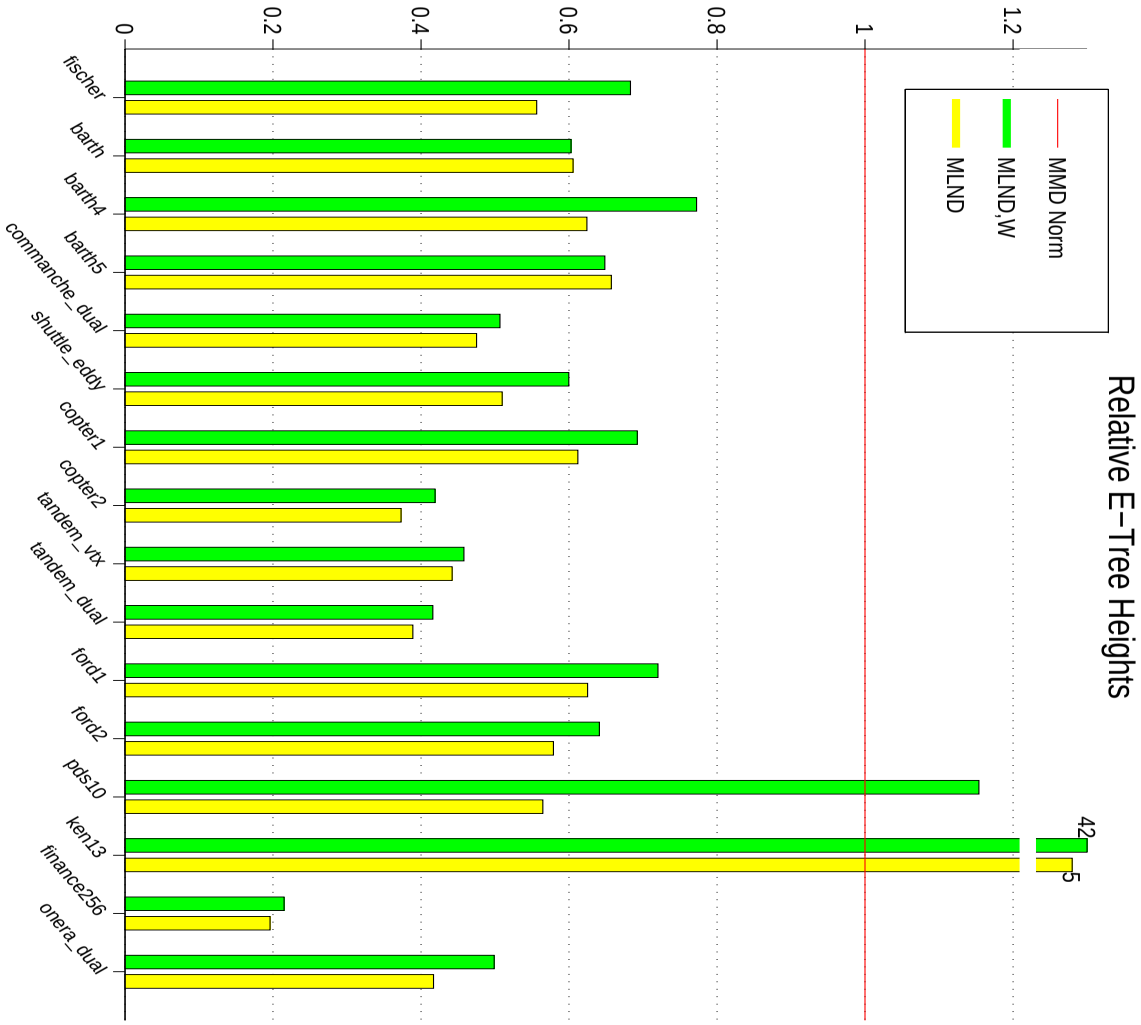

Figure 3: Elimination tree heights of multilevel ordering algorithms.

The results show that for the majority of the problems, the MLND algorithms are capable of reducing the work and elimination tree heights over the MMD algorithm. We first discuss the performance of the structural and fluid dynamics problems. For these problems the MLND elimination tree heights are less than 60% of the MMD values, and this leads to shorter critical path lengths and higher parallelism in the factorization. The work is also smaller with the MLND orderings for these problems with the exception of the small COMMANCHE_DUAL problem. The MLND and MLNDW algorithms perform relatively similarly for these problems.

Now we discuss the performance of these ordering algorithms on the linear programming problems (lp's). These problems are obtained by forming the matrices AA^T, where A is the constraint matrix of the linear program, as in an interior point method for solving the lp's. For FINANCE256, the MLND variants perform substantially better than MMD; this behavior was also observed for the spectral nested dissection algorithm (SND), and it can be explained in terms of the structure of the underlying linear program. Roughly the structure of this problem is that of a complete binary tree, with each node of the tree corresponding to a subgraph. A small set of edges joins each subgraph to its parent in the tree. The nested dissection algorithms partition the binary tree at the root, and thus compute good separators. The MMD algorithm orders an independent set of low degree vertices from all over the tree, destroys the binary tree structure, and consequently computes larger separators.

For the problems PDS10 and especially KEN13, the nested dissection algorithms are outperformed by the MMD algorithm, and the MLND algorithm performs strikingly better than the MLNDW algorithm. A careful examination of KEN13 shows it to have considerable structure: it consists of 169 subgraphs of 169 vertices each, joined together by 71 independent vertices. We are currently studying the behavior of these algorithms on this example. These examples should convince the reader that we need more experience with these algorithms to understand their performance on different classes of problems.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi, *Cutting down on fill using nested dissection: provably good elimination orderings*, in Graph

Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., vol. 56 of IMA Volumes in Mathematics and its Applications, Springer Verlag, 1993, pp. 31–55.

[2] N. Alon and V. Milman, λ_1, *isoperimetric inequalities for graphs, and superconcentrators*, J. of Combin. Theory, Series B, 38 (1985), pp. 73 – 88.

[3] C. J. Alpert and A. Kahng, *Recent directions in netlist partitioning: A survey*, tech. report, Computer Science Department, University of California at Los Angeles, 1995.

[4] K. S. Arun and V. B. Rao, *Constructive heuristics and lower bounds for graph partitioning based on a principal components approximation*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 991–1015.

[5] C. Ashcraft and J. W. H. Liu, *Using domain decomposition to find graph bisectors*. Preprint, 1995.

[6] B. Aspvall and J. R. Gilbert, *Graph coloring using eigenvalue decomposition*, SIAM J. Alg. Disc. Meth., 5 (1984), pp. 526 – 538.

[7] S. T. Barnard, A. Pothen, and H. D. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, J. Numerical Linear Algebra with Applications, 2 (1995), pp. 317–334.

[8] S. T. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurreny: Practice and Experience, 6 (1994), pp. 101–117.

[9] E. R. Barnes, *An algorithm for partitioning the nodes of a graph*, SIAM J. Alg. Disc. Meth., 3 (1982), pp. 541–550.

[10] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, 1993. Second edition.

[11] B. Bollobás, *Graph Theory: An introductory course*, Springer Verlag, 1979.

[12] R. B. Boppana, *Eigenvalues and graph bisection: an average case analysis*, in 28th Annual Symp. Found. Comp. Sci, 1987, pp. 280–285.

[13] T. N. Bui, J. Fukuyama, and C. Jones, *The planar vertex separator problem: Complexity and algorithms*. Manuscript, 1994.

[14] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Inf. Process. Letters, 42 (1992), pp. 153–159.

[15] T. N. Bui and C. Jones, *A heuristic for reducing fill-in in sparse matrix factorization*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.

[16] T. N. Bui and B. R. Moon, *A genetic algorithm for a special class of the quadratic assignment problem*, in The Quadratic Assignment and Related Problems, P. Pardalos and H. Wolkowicz, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1993.

[17] T. F. Chan, P. Ciarlet, Jr., and W. K. Szeto, *On the near optimality of the recursive spectral bisection method for graph partitioning.* Manuscript, Feb. 1993.

[18] T. F. Chan, J. R. Gilbert, and S.-H. Teng, *Geometric spectral bisection.* Preprint, 1994.

[19] T. F. Chan and T. P. Mathew, *Domain decomposition algorithms*, Acta Numerica, (1994), pp. 61–143.

[20] N. Chrisochoides, E. Houstis, and J. Rice, *Mapping algorithms and software environment for data parallel PDE iterative solvers*, J. Parallel and Distributed Computing, (1994). To appear.

[21] N. Chrisochoides, N. Mansour, and G. Fox, *Performance evaluation of data mapping algorithms for parallel single-phase iterative PDE solvers.* Preprint, 1993.

[22] E. C. Chu, J. A. George, J. W. Liu, and E. G. Ng, *User's guide for SPARSPAK-A: Waterloo sparse linear equations package*, Tech. Report CS-84-36, Computer Science, University of Waterloo, Ontario, Canada, 1984.

[23] P. Ciarlet, Jr. and F. Lamour, *Spectral partitioning methods and greedy partitioning methods: A comparison on finite element graphs*, Tech. Report 94-9, Department of Mathematics, University of California at Los Angeles, April 1994.

[24] P. Ciarlet, Jr., F. Lamour, and B. F. Smith, *On the influence of the partitioning scheme on the efficiency of overlapping domain decomposition methods.* Fifth Symposium on the Frontiers of Massively Parallel Computation, 1995.

[25] D. Cvetkovic, M. Doob, and H. Sachs, *Spectra of Graphs*, Academic Press, New York, 1980.

[26] D. M. Cvetkovic, M. Doob, I. Gutman, and A. Torgasev, *Recent Results in the Theory of Graph Spectra, Annals of Dicrete Math.*, vol. 36, North Holland, 1988.

[27] L. Dagum, *Automatic partitioning of unstructured grids into connected components.* Preprint, 1995.

[28] R. Das, D. J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy, *The design and implementation of a parallel unstructured Euler solver using software primitives*, in Proceedings of AIAA 30th Aerospace Sciences Meeting, 1992. Paper AIAA-92-0562.

[29] W. E. Donath and A. J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM Journal of Research and Development, 17 (1973), pp. 420–425.

[30] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[31] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM TOMS, 15 (1989), pp. 1 – 14.

[32] J. Falkner, F. Rendl, and H. Wolkowicz, *A computational study of graph partitioning*, Math. Programming, (1994). To appear.

[33] C. Farhat, S. Lanteri, and H. D. Simon, *TOP/DOMDEC: a software tool for mesh partitioning and parallel processing.* J. Comput. Sys. Engrg. (to appear), 1993.

[34] C. Farhat and F. X. Roux, *Implicit parallel processing in structural mechanics*, Computational Mechanics Advances, 2 (1994), pp. 1–124.

[35] C. Fiduccia and R. Mattheyses, *A linear time heuristic for improving network partitions*, in ACM - IEEE 19th Design Automation Conference, Las Vegas, IEEE Press, 1982, pp. 175 – 181.

[36] M. Fiedler, *Algebraic connectivity of graphs*, Czech. Math. J., 23 (1973), pp. 298–305.

[37] ——, *A property of eigenvectors of non-negative symmetric matrices and its application to graph theory*, Czech. Math. J., 25 (1975), pp. 619–633.

[38] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1979.

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

[40] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[41] J. A. George and J. W. H. Liu, *The evolution of the minimum degree algorithm*, SIAM Rev., 31 (1989), pp. 1–19.

[42] J. A. George and A. Pothen, *Analysis of the spectral approach to envelope reduction via a quadratic assignment formulation.* To appear in SIAM J. Matrix Anal. Applic., 1995.

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

[44] J. R. Gilbert, G. L. Miller, and S.-H. Teng, *Geometric mesh partitioning: Implementation and experiments*, Tech. Report CSL–94–13, Xerox Palo Alto Research Center, 1994.

[45] J. R. Gilbert and E. Zmijewski, *A parallel graph partitioning algorithm for a message-passing multiprocessor*, International Journal of Parallel Programming, 16 (1987), pp. 427–449.

[46] G. H. Golub and C. F. V. Loan, *Matrix Computations*, Johns Hopkins University Press, 1989. Second edition.

[47] S. Guattery and G. Miller, *On the performance of spectral graph partitioning methods*, in 6th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, January 1995, ACM-SIAM, pp. 233–242.

[48] L. Hagen and A. Kahng, *New spectral methods for ratio cut partitioning and clustering*, IEEE Transactions on Computer-Aided Design, 11 (1992), pp. 1074–1085.

[49] K. M. Hall, *An r-dimensional quadratic placement algorithm*, Management Science, 17 (1970), pp. 219–229.

[50] S. Hammond, *Mapping unstructured grid computations to massively parallel computers*, PhD thesis, Rensellaer Polytechnic Institute, Troy, NY, 1992. RIACS Report 92-14.

[51] M. T. Heath, E. G. Y. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Rev., 33 (1991), pp. 420–460.

[52] M. T. Heath and P. Raghavan, *A Cartesian parallel nested dissection algorithm*, SIAM J. Matrix Anal. Appl., (1995).

[53] B. Hendrickson and R. Leland, *The CHACO user's guide*, Tech. Report 93–2339, Sandia National Labs, Albuquerque NM, 1993.

[54] ——, *Multidimensional load balancing*, Tech. Report 93–0074, Sandia National Labs, Albuquerque NM, 1993.

[55] ——, *A multilevel algorithm for partitioning graphs*, Tech. Report 93–1301, Sandia National Labs, Albuquerque NM, 1993.

[56] B. Hendrickson and R. Leland, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM J. Stat. Comput., 16 (1995), pp. 452–469.

[57] D. C. Hodgson and P. K. Jimack, *Efficient mesh partitioning for parallel PDE solvers on distributed memory machines*, Tech. Report 93-1, School of Computer Studies, University of Leeds, 1993.

[58] S. H. Hsieh, G. H. Paulino, and J. F. Abel, *Recursive spectral algorithms for automatic domain partitioning in parallel finite element analysis*. Preprint, Civil Engineering, Cornell University, July 1993.

[59] Z. Johan, *Data parallel finite element techniques for large-scale Computational Fluid Dynamics*, PhD thesis, Stanford University, 1992.

[60] Z. Johan, K. K. Mathur, S. L. Johnsson, and T. J. R. Hughes, *Parallel implementation of recursive spectral bisection on the Connection Machine CM-5 system*, Tech. Report 07-94, Center for Research in Computing Technology, Harvard University, 1994.

[61] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon, *Optimization by simulated annealing: An experimental evaluation, Part I, Graph Partitioning*, Operations Research, 37 (1989), pp. 865–892.

[62] C. Jones, *Vertex and edge partitions of graphs*, PhD thesis, The Pennsylvania State University, 1992.

[63] G. Karypis and V. Kumar, *Analysis of multilevel graph partitioning*, Tech. Report 95-037, Computer Science Department, University of Minnesota, 1995.

[64] ——, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Report 95-035, Computer Science Department, University of Minnesota, 1995.

[65] ——, *Parallel multilevel graph partitioning*, Tech. Report 95-036, Computer Science Department, University of Minnesota, 1995.

[66] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical J., 49 (1970), pp. 291–307.

[67] D. E. Keyes, *Domain decomposition: A bridge between nature and parallel computers*, in Adaptive, Multilevel and Hierarchical Computational Strategies, A. K. Noor, ed., Amer. Soc. Mech. Eng., New York, 1992, pp. 293–334.

[68] D. E. Keyes, Y. Saad, and D. G. Truhlar, eds., *Domain-based Parallelism and Problem Decomposition Methods in Science and Engineering*, SIAM, 1995.

[69] G. Kumfert and A. Pothen, *A multilevel nested dissection algorithm*. Unpublished work, 1995.

[70] C. A. Leete, B. W. Peyton, and R. F. Sincovec, *Toward a parallel recursive spectral bisection mapping tool*, in Proceedings of the Sixth SIAM Conference on Parallel Processing, SIAM, 1993, pp. 923–928.

[71] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proc. of the 29th Annual Symposium on Foundations of Computer Science, IEEE, 1988, pp. 422–431.

[72] P. J. LeTallec, *Domain decomposition methods in computational mechanics*, Computational Mechanics Advances, 2 (1994), pp. 121–220.

[73] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1146–1173.

[74] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[75] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

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

[77] ——, *The minimum degree ordering with constraints*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 198–219.

[78] ——, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[79] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Automatic mesh partitioning*, in Graph Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer Verlag, 1993, pp. 57–84. The IMA Volumes in Mathematics and its Applications, Volume 56.

[80] B. Mohar, *The Laplacian Spectrum of Graphs*, J. Wiley, New York, 1991, pp. 871–898.

[81] B. Mohar and S. Poljak, *Eigenvalues in combinatorial optimization*. Preprint, 1992.

[82] B. Nour-Omid, A. Raefsky, and G. Lyzenga, *Solving finite element equations on concurrent computers*, in Parallel Computations and their Impact on Mechanics, A. K. Noor, ed., New York, 1986, American Soc. of Mech. Eng., pp. 209 – 227.

[83] J. T. Oden, A. Patra, and Y. Feng, *Domain decomposition for adpative hp finite element methods*, in Domain Decomposition Methods in Science and Engineering, D. E. Keyes and J. Xu, eds., vol. 180 of Contemporary Mathematics, American Mathematical Society, 1994, pp. 295–301.

[84] C. Özturan, *Distributed environment and load balancing for adaptive unstructured meshes*, PhD thesis, Rensellaer Polytechnic Institute, Troy, NY, 1995.

[85] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[86] A. Pothen, *An analysis of spectral graph partitioning via quadratic assignment problems*, in Domain Decomposition Methods in Science and Engineering, D. E. Keyes and J. Xu, eds., vol. 180 of Contemporary Mathematics, American Mathematical Society, 1994, pp. 105–110.

[87] A. Pothen and C.-J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Transactions on Mathematical Software, 16 (1990), pp. 303–324.

[88] A. Pothen, E. Rothberg, H. D. Simon, and L. Wang, *Parallel sparse Cholesky factorization with spectral nested dissection ordering*, in Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, J. G. L. et al., ed., SIAM, 1994, pp. 418–422.

[89] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.

[90] A. Pothen, H. D. Simon, and L. Wang, *Spectral nested dissection*, Tech. Report CS-92-01, Computer Science, Pennsylvania State University, University Park, PA, 1992.

[91] A. Pothen, L. Wang, H. Simon, and S. Barnard, *Towards a fast implementation of spectral nested dissection*, in Proceedings of Supercomputing '92, Washington, 1992, IEEE Computer Society Press, pp. 536 – 538.

[92] F. Rendl and H. Wolkowicz, *A projection technique for partitioning the nodes of a graph*, Annals of Operations Research, (1994). This paper was written in 1990. To appear in the special issue devoted to the Symposium on Applied Mathematical Programming and Modeling, Budapest, Jan. 1993.

[93] D. Roose and R. VanDriessche, *Distributed memory parallel computers and Computational Fluid Dynamics*, Tech. Report TW 186, Department of Computer Science, Katholike Universiteit Leuven, Belgium, 1993.

[94] H. Schramm and J. Zowe, *A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results*, SIAM J. Opt., 2 (1992), pp. 121–152.

[95] M. S. Shephard, J. E. Flaherty, H. L. DeCougny, C. Özturan, C. L. Bottasso, and M. Beall, *Parallel automated adaptive procedures for unstructured meshes*, in Parallel Computing in CFD, Advisory Group for Aerospace Research and Development (AGARD), Neuilly-Sur-Seine, France, 1995. Report R-807.

[96] H. D. Simon, *Partitioning of unstructured problems for parallel processing*, Computing Systems in Engineering, 2 (1991), pp. 135–148.

[97] B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Algorithms for Elliptic Partial Differential Equations*, Cambridge Univ. Press, 1995.

[98] P. Suaris and G. Kedem, *An algorithm for quadrisection and its application to standard cell placement*, IEEE Transactions on Circuits and Systems, 35 (1988), pp. 294–303.

[99] R. Van Driessche and D. Roose, *A graph contraction algorithm for the fast calculation of the Fiedler vector of a graph*, in Proceedings of the Seventh SIAM Conference on Parallel Computing for Scientific Computing, 1995, pp. 621–626.

[100] D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings, and O. Zone, *A retrofit based methodology for the fast generation and optimization of large-scale mesh partitions*, Tech. Report CAS-94-18, Center for Aerospace Structures, University of Colorado, Boulder, 1994.

[101] R. VanDriessche and D. Roose, *An efficient spectral bisection algorithm for dynamic load balancing*, Tech. Report TW 215, Department of Computer Science, Katholike Universiteit Leuven, Belgium, 1994.

[102] V. Venkatakrishnan, H. D. Simon, and T. Barth, *A MIMD implementation of a parallel Euler solver for unstructured grids*, J. Supercomputing, 6 (1992), pp. 117–127.

[103] C. Walshaw and M. Berzins, *Dynamic load balancing for PDE solvers on adaptive unstructured meshes*, Tech. Report 92-32, School of Computer Studies, University of Leeds, 1992.

[104] L. Wang, *Spectral Nested Dissection Orderings for Parallel Sparse Matrix Factorization*, PhD thesis, Computer Science Department, The Pennsylvania State University, 1994.

[105] R. D. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency, 3 (1991), pp. 457–481.