A Graph Based Davidson Algorithm for the Graph Partitioning Problem

Michael Holzrichter
Sandia National Laboratories
Albuquerque, NM 5800, USA

and

Suely Oliveira
Dept. of Computer Science, The University of Iowa,
Iowa City, IA 52242, USA

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

The problem of partitioning a graph such that the number of edges incident to vertices in different partitions is minimized, arises in many contexts. Some examples include its recursive application for minimizing fill-in in matrix factorizations and load-balancing for parallel algorithms. Spectral graph partitioning algorithms partition a graph using the eigenvector associated with the second smallest eigenvalue of a matrix called the *graph Laplacian*. The focus of this paper is the use graph theory to compute this eigenvector more quickly.

Keywords: Graph Partitioning Algorithms, Spectral Algorithms, Fiedler Vector, Eigenvalue solvers

1. Introduction

The problem of partitioning a graph such that the number of edges incident to vertices in different partitions is minimized arises in many contexts. For example, the nested dissection algorithm [13] uses a solution to the graph partitioning problem to find permutations of sparse matrices which reduce fill-in. Graph partitioning has been used in many other applications such as VLSI design and load-balancing for parallel algorithms.

Spectral algorithms partition a graph based on the *Fiedler vector*, which is the second smallest eigenvector of a matrix, called the "graph Laplacian", associated with a graph. Finding this eigenvector represents the majority of the time required by spectral partitioning techniques.

Fiedler [11, 12] did pioneering work on the relationship between the properties of a graph and the the *Fiedler vector*. Pothen, Simon, and Liou [26], explored the usage of the Fiedler vector for partitioning graphs. Hendrickson and Leland [17] generalized spectral bisection to perform quadrasection and octasection by using the third and fourth smallest eigenpairs of the graph Laplacian. Spielman and Teng [30] present an upper bound on the Fiedler value of bounded degree d-dimensional graphs, and they relate this upper bound to the number of edges cut by a Fiedler cut.

In this paper we apply graph theory to improve the computation of the Fiedler vector. In our new approaches the eigensolver is the Davidson algorithm, which is a subspace method allowing the use of preconditioners. The use of the Davidson algorithm for graph partitioning has not previously appeared in the literature. We modify the original Davidson algorithms to incorporate information about the graph in the form of a preconditioner. In addition, we use information about the graph to develop efficient implementations. In other words, the original Davidson algorithm is modified in three ways: First, a series of coarse graphs is used as a framework for a multilevel preconditioner. Second, a favorable initial subspace for the Davidson algorithm is obtained by running the Davidson algorithm on the next coarser graph. Third, the series of coarse graphs were used to permute the input matrix to increase the concentration of the non-zero entries along the diagonal (this increased temporal and spatial locality of reference while computing the Fiedler vector).

The Fiedler vector has proven to be useful for more than graph partitioning. Recent work uses the Fiedler vector to impose a total reorder rather than just a partition. This has been found to be useful for two closely related problems, envelope reduction and seriation. Envelope reduction [2, 14] attempts to minimize the maximum distance of non-zero elements from the diagonal. Atkins, Boman, and Hendrickson [1] discuss the seriation problem, which is to reorder a set of elements such that closely coupled elements appear as near as possible to each other in the new ordering.

2. Spectral Methods

2.1. The graph Laplacian

Every undirected graph can be represented by an adjacency matrix. Similarly, every graph can be associated with a similar matrix, called its *graph Laplacian*. Spectral partitioning methods partition a graph using the *second smallest eigenvector* of its graph Laplacian (also known as Fiedler vector). Many properties of the graph Laplacian are discussed in [6, 32].

Let A be the adjacency matrix associated with a graph and D be a matrix such that all entries are zero except along the diagonal where $d_{i,i}$ is equal to $d(v_i)$, the degree of vertex v_i. The *graph Laplacian* L is defined as $D - A$, i.e., in other words the elements of L are given by $l_{i,j} = d(v_i)$ if $i = j$, -1 if $i \neq j$, and the graph has an edge (v_i, v_j), and 0 otherwise.

There is a one-to-one correspondence between vertices and elements of the Fiedler vector. A graph can be bisected by finding the median element of the Fiedler vector and then assigning vertices to one partition or another based on whether the corresponding element in the Fiedler vector is larger or smaller than the median. A similar process is used for other types of cuts such as "ratio cuts" by choosing another element instead of the median.

2.2. The Davidson's Eigensolver

Many algorithms have been developed for computing eigenvalues and eigenvectors. General surveys of these techniques can be found in [8, 15, 25, 28]. In our work we introduce Davidson-type algorithms [4, 7, 24] as *a basis for graph partitioning algorithms*. Furthermore, we use graph theory to incorporate information about the problem into the Davidson eigensolver. Davidson-type algorithms are iterative subspace methods for finding a limited number of eigenpairs. They build up a set of orthogonal vectors which span nested subspaces of increasing dimension. Each iteration adds an additional basis vector, increasing the dimension of the subspace by one. A new basis vector is created by multiplying the residual by preconditioner M_λ and orthogonalizing the result against the existing basis vectors. If the identity were used as the preconditioner, Davidson's subspaces would be Krylov spaces and Davidson's algorithm would be equivalent to the Lanczos algorithm with re-orthogonalization. The Davidson algorithm, like the Arnoldi algorithm, works well for non-symmetric matrices while the Lanczos algorithm is for symmetric matrices only. The pseudo-code for Davidson's method for computing the smallest eigenpair follows:

Algorithm 1 – Basic Davidson's Algorithm

Given matrix C, iteration limit m, and convergence factor ϵ, compute eigenpair λ, u.

1. Choose an initial vector v_1. Normalize v_1 and make the result the sole column of V_1.
2. For $j = 1, 2, \ldots, m$ or convergence do:
   (a) Compute $w := Cv_j$.
   (b) Compute $V_j^T w$, and make this the last column and row of S_j where $S_j = V_j^T C V_j$.
   (c) Compute the smallest eigenpair $\tilde{\lambda}$, y of S_j.
   (d) Form the Ritz vector $\tilde{u} := V_j y$.
   (e) Compute the residual $r := C\tilde{u} - \tilde{\lambda}\tilde{u}$; exit loop if $\|r\| < \epsilon$.
   (f) Compute $t := M_\lambda r$.
   (g) Orthogonalize t against the columns of V_j to form v_{j+1}. Append v_{j+1} to V_j to form V_{j+1}.
3. Return eigenpair λ, u where $\lambda = \tilde{\lambda}$ and $u = \tilde{u}$.

2.3. Multilevel Graph Bisection

Recently multilevel algorithms for partitioning graphs have been an area of active research [3, 16, 21]. Multilevel algorithms work with the graph at multiple levels of granularity. Increasing levels of granularity are obtained by combining adjacent vertices into "composite" vertices at the next coarser level of granularity. This maps the graph into another graph with fewer vertices. Connectivity of the original graph is preserved by ensuring that two vertices at level k are connected by an edge if, and only if, there exists one or more edges between vertices at the next finer level which were combined to form the vertices at the coarse level.

The process of coarsening a graph by combining pairs of adjacent vertices is repeated until a graph is obtained with sufficiently few vertices that another partitioning algorithm can quickly find a high-quality partition. An initial partition for the coarsest graph is interpolated back onto the graph at the next finer level. After interpolation, the partition may be improved using a local refinement algorithm and the interpolation is repeated until the original (finest) level of the graph has been reconstructed.

The basic outline of this class of multilevel partitioning algorithms is 1) Coarsen the graph until the number of vertices is sufficiently small. 2) Partition the graph at the coarsest level, and 3) Uncoarsen and refine the partition level by level until a partition of the original graph is obtained.

There are a variety of choices for each of these steps. Karypis and Kumar explore many choices in [21]. To construct the next coarser graph G_{i+1} from graph G_i, a maximal matching (maximal with respect to inclusion, not size) is found for G_i. A maximal matching for a graph is not unique. Karypis and Kumar construct maximal matchings by visiting the vertices of a graph in some order. If the vertex is adjacent to one or more unmatched vertices, then one of the unmatched neighbors is chosen to be paired with the visited vertex. If the vertex has no unmatched vertices, it is simply copied to the next coarser graph. Once all the vertices have been visited, a maximal matching will have been found. The coarse graph G_{i+1} can be constructed from the fine graph G_i by combining the matched vertices.

Provided the coarsest graph is sufficiently small, a wide range of partitioning methods, including spectral partitioning, can quickly find a high quality partition of the coarse graph. The time required to partition the coarse graph becomes a minor consideration and the partitioning technique can be chosen based on the characteristics of the partition produced. Variations of the Kernighan-Lin algorithm [22] are widely used for this step. Spectral techniques (usually based on the Lanczos algorithm) have been employed to partition the coarsest graph.

When a partition is interpolated onto the next finer graph, the result is a partition which is not optimal for that graph. That partition is refined after interpolation. A variation of Kernighan-Lin called "Boundary Kernighan-Lin" (BKL) can be used to refine the graph. BKL is efficient during the refinement phase because it restricts the pairs of vertices considered for swapping to those which lie along the boundary between the two partitions. This significantly speeds the refinement step.

3. New Methods

In the previous section, we mentioned that traditional spectral partitionings (such as Lanczos algorithm for example) are only applied to the coarsest representation of the graph. Because our algorithms are cheaper than the Lanczos, they make it possible to perform spectral partitioning of the original graph. In addition, they are not coupled with other heuristic algorithms. Our methods are purely spectral methods.

Multilevel algorithms find a partition for an input graph G_0 by working with additional graphs $G_1, G_2, \ldots G_n$, constructed by successive coarsening of G_0. Previous multilevel graph partitioning algorithms worked by transforming an initial partition of the coarsest graph G_n into a partition for G_0 by interpolation onto successively finer graphs followed by local refinement. The key feature of the new approaches presented here is that, rather than partitioning the coarsest graph G_n, information about the graph's structure provided by $\{G_0, G_1, \ldots, G_n\}$ is used to accelerate the computation of the Fiedler vector for the *input* graph.

Due to the method used to construct them, vertices of a coarse graph G_j represent highly connected regions of G_0. The size of the region of G_0 represented by a vertex of G_j increases with j, i.e. increases with the coarseness of the graph. Thus, the series of graphs $\{G_0, G_1, \ldots, G_n\}$ are one kind of representation of the graph's structure.

Our new algorithms use the information about the graph's structure represented by $\{G_0, G_1, \ldots, G_n\}$ in novel ways. The first two new methods (Sections 3.1 and 3.2) we will present achieves the goal of reducing the number of iterations required to find the Fiedler vector. The third new method (Section 3.3) enhances the implementation of the eigensolver by reordering the data structures in memory to increase spatial and temporal locality, improving the effectiveness of the computer's cache memory. We call the resulting method when one or more of these algorithms are used and applied to the input graph, "Multilevel Davidson Algorithm" (MDA).

3.1. Preconditioned Davidson Algorithm (PDA)

One property of graph Laplacian matrices is that the smallest eigenvalue is always 0 and the corresponding eigenvector is the vector $(1, 1, \ldots, 1)^T$. Based on this *a priori* knowledge, the initial set of basis vectors is augmented with an additional basis vector - the unit vector in the direction of $(1, 1, \ldots, 1)^T$.

Due to the size of L, solving $Lt_i = r_i$ can be expensive. We developed an algorithm for solving this system based on graphs $\{G_0, G_1, \ldots, G_n\}$. This gives us a framework for a multilevel preconditioner for solving $Lt_i = r_i$.

Multilevel iterative methods have proven to be effective for solving systems of linear equations arising from PDE's [5, 23]. While the traditional multilevel approach for solving elliptic PDE's obtains smaller problems by coarsening the spatial discretization, our Preconditioned Davidson Algorithm implements a preconditioner which uses $\{G_0, G_1, \ldots, G_n\}$ as its multilevel framework. This preconditioner can be used on irregular matrices arising from unstructured graphs where the usual coarsening process is ill-defined or not applicable.

Let G_0, be the input graph which has been successively coarsened to form G_1 to G_n. Let L_j be the graph Laplacian associated with graph G_j and r_0 be the residual at the end of the i^{th} iteration. (The iteration index i is implicit since one application of the preconditioner begins and ends in the same iteration of the Davidson algorithm.) The restriction of r_0 onto graph G_j is r_j. The basic idea behind the multilevel preconditioner is to use relaxation to find an approximate solution t_j to $L_j t_j = r_j$. Projecting this solution onto the next finer graph G_{j-1} provides a good starting point for performing relaxation to find t_{j-1} which is a solution to $L_{j-1} t_{j-1} = r_{j-1}$.

The new multilevel preconditioner, expressed as pseudocode, follows:

Algorithm 2 – Multilevel Preconditioner for Laplacian System

Given residual vector r_0, graphs $\{G_0, G_1, \ldots, G_n\}$, graph Laplacians $\{L_0, L_1, \ldots, L_n\}$, and relaxation count parameters n_1 and n_2, compute vector t_0, which is an approximate solution to $L_0 t_0 = r_0$.

1. Compute $\{r_1, \ldots, r_n\}$ by successively restricting r_0 to graphs $G_1, G_2, \ldots, G_n$.
2. Find t_n which solves $L_n t_n = r_n$.
3. For $i = n-1, n-2, ...0$ do
   (a) Create t_i by interpolating t_{i+1} onto graph G_i.
   (b) Set $y_i = r_i$.
   (c) For $j = i, i+1, \ldots, n$ do
      i. Relax n_1 times the system $L_j t_j = y_j$ using Gauss-Seidel.
      ii. Form y_{j+1} by computing the residual $L_j t_j - y_j$ and restricting onto graph G_{j+1}.
   (d) Find t_n which solves $L t_n = y_n$.
   (e) For $j = n-1, n-2 \ldots, i$ do
      i. Interpolate t_{j+1} onto graph G_j and sum with existing t_j.
      ii. Relax n_2 times the system $L_j t_j = y_j$ using Gauss-Seidel.
4. Return t_0 as an approximate solution to $L_0 t_0 = r_0$.

Figure 1 illustrates our multilevel preconditioner for a graph which has been coarsened to four levels. Figure 1 shows the path though the different levels of graphs and the operations performed along the way. The process is very similar to full multigrid V-cycles, but differs by using graph theory to define the restriction and interpolation operations. The circles, squares, triangles, and hexagons represent an operation using the graph at that level as input. The action represented by the different shapes are:

- **Circle -** solve $L_n t_n = y_n$ on the coarsest graph. This is steps 2 and 3-d in the pseudocode.
- **Square -** interpolate t_{i+1} found for graph G_{i+1} onto graph G_i to form t_i. This is steps 3-a and 3-b in the pseudocode.
- **Triangle -** interpolate t_{j+1} onto graph G_j and sum with existing t_j then relax $L_j t_j = y_j$ n_2 times using Gauss-Seidel. This is step 3-e in the pseudocode.
- **Hexagon -** relax $L_j t_j = y_j$ n_1 times using Gauss-Seidel then restrict the residual $L_j t_j - u_j$ onto graph G_{j+1} to form y_{j+1}. This is step 3-c in the pseudocode.

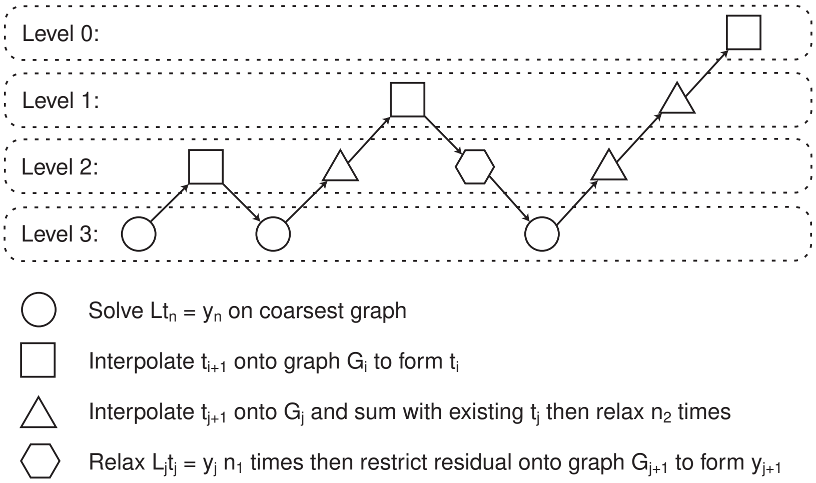

Figure 1: Operation of multilevel preconditioner for four graph levels

3.2. Nested Davidson Algorithm (NDA)

The Nested Davidson Algorithm generalizes the Davidson algorithm by allowing the use of multiple vectors for the initial subspace. Advantageous basis vectors for G_0 are obtained by performing Davidson on G_1, the next coarser graph and interpolating linear combinations of the basis vectors for G_1 onto G_0 then reorthogonalizing. The assumption is that basis vectors for G_1 will yield favorable basis vectors when interpolated onto G_0 such that the distance between the Fiedler vector and its interpolation into the subspace is small. Moreover, multiple bases allows more degrees of freedom when approximating the Fiedler vector.

The same approach can be used to accelerate the Davidson algorithm on G_1 making NDA a recursive algorithm. The recursion terminates with the coarsest graph, G_n. Linear combinations of the resulting basis vectors are interpolated onto the next finer graph G_{n-1} to create an initial subspace as a starting point for NDA on G_{n-1}. The process works its way up the hierarchy of graphs until the Fiedler vector for the finest graph G_0 has been computed.

The Nested Davidson Algorithm and Barnard and Simon's Multilevel Spectral Bisection (MSB) [3] both find the Fiedler vector for finer and finer graphs using the interpolation of the result from the next coarser graph as a starting point for the current graph. While MSB uses the Rayleigh Quotient Iteration to refine the newly interpolated Fiedler vector, NDA interpolates basis vectors onto the finer graph and uses them as a set of basis vectors for the initial subspace of our PDA algorithm. The Nested Davidson Algorithm modifies the Davidson algorithm in the following way:

First, before the Fiedler vector is computed for G_j, NDA recursively computes the Fiedler vector of the next coarser graph, G_{j+1}. The recursion yields l basis vectors for G_{j+1}. These basis vectors are interpolated onto G_j and then reorthonormalized. These vectors become the basis for the initial subspace for the Davidson algorithm applied to the graph Laplacian of G_j. The l basis vectors passed to the caller are weighted sums of the k basis vectors for the subspace constructed at the current level of recursion.

Let $\{a_0, a_1, \ldots a_k\}$ be the weights of the expansion of the Fiedler vector at level $j+1$ in terms of the basis vectors for that level. Let $s(i)$ be a function which gives the index of the weight with the i^{th} largest *magnitude*. l vectors $\{b_0, b_1, \ldots b_{l-1}\}$ are formed from the k basis vectors $\{v_0, v_1, \ldots v_{k-1}\}$ according to the formula:

$$b_i = \sum_{j=0}^{p} a_{s(i+l*j)} v_{s(i+l*j)} \quad \text{where} \quad p = \begin{cases} \lfloor k/l \rfloor + 1 & \text{if } i < k \bmod l \\ \lfloor k/l \rfloor & \text{otherwise.} \end{cases}$$

This creates l vectors from the k basis vectors such that the Fiedler vector is a linear combination of these l new vectors. These vectors $\{b_0, b_1, \ldots b_{l-1}\}$ are interpolated onto G_j and reorthogonalized.

The pseudocode for the Nested Davidson Algorithm is:

Algorithm 3 – Nested Davidson Algorithm

Given graphs $\{G_i, G_{i+1}, \ldots, G_n\}$, graph Laplacians $\{L_i, L_{i+1}, \ldots, L_n\}$, iteration limit m and maximum dimension of initial subspace l, compute eigenvector x and B, whose l columns are linear combinations the columns of V such that x lies in the subspace spanned by the columns of B.

1. Set $v_0 := (1, \ldots, 1)/\|(1, \ldots, 1)\|$.
2. Set $S_{0,0} := v_0^T L_j v_0$.
3. If current graph is G_n then
   (a) Choose arbitrary v_1 and normalize.

   Otherwise

   (b) *nested_davidson*(G_{j+1}, B)

   (c) For $k = 1$ to l do:
      i. Interpolate b_{k-1} onto graph G_j and orthogonalize against columns of V_{k-1} to form v_k. Append v_k to V_{k-1} as the last column to form V_k.
      ii. Compute $V_k L_j v_k$ and make the result the last row and column of S.
4. For $k = l, 2, \ldots, m + l - 1$ or convergence do:
   (a) Perform Davidson iterations.
5. Return v_k as x. Construct l vectors from linear combinations of the k current basis vectors and store them as the columns of B.

More details about NDA are presented in [19, 18].

3.3. Multilevel Graph Reordering (MGR)

Graph Laplacian matrices are stored in memory using large, irregular data structures. These data structures may result in memory access patterns with poor temporal and spatial locality, thereby harming performance. Multilevel Graph Reordering (MGR) is a heuristic we use to reorder the storage of graph data structures to improve the benefit of cache memories when performing the PDA and NDA algorithms.

Douglas [10] studied ways to modify multigrid algorithms so that their memory access patterns take better advantage of cache memory. Douglas' solution is predicated upon regularity in the discretization of the problem. The PDA and NDA algorithms do not assume any regularity in the structure of the input matrix so Douglas' techniques are not applicable. Toledo [31] discussed locality of reference when performing LU decomposition. MGR addresses a problem distinct from those considered by Douglas, namely the locality behavior of PDA and NDA when used on graphs of unknown or irregular structure. Pothen and Ye have presented preliminary results for reordering data access in a irregular mesh for an Euler Equation solver [27]. Their reordering data accesses include Cuthill-McKee, Sloan, Nested Dissection, and space-filling-curve algorithms and are disctinct from the method we present below.

MGR attempts to enhance locality by permuting the graph Laplacian matrices in a manner which concentrates the non-zero entries near the diagonal. Concentrating non-zero elements along the diagonal improves locality of memory references during relaxation, which is the major time consuming step in our calculation of the Fiedler vector.

In addition to permuting the matrix, the data structures are rearranged in memory. This was necessary because the random order in which vertices are visited during coarsening resulted in data structures with non-contiguous indices stored contiguously in memory. The order of edges in the edge list were sorted into increasing order also. As a result, vertices with contiguous indices were stored in data structures with contiguous memory addresses.

The graphs $\{G_0, G_1, \ldots, G_n\}$ provide the means to find a permutation which increases the density of non-zero elements along the diagonal quickly and efficiently.

Our MGR implementation extended $\{G_0, G_1, \ldots G_n\}$ with additional coarse graphs until a graph with a single vertex was obtained. A tree was imposed on the vertices. (The edges of the graphs were ignored during the relabeling process.) The children of a vertex in graph G_j are the vertices of G_{j-1} which were combined to form that vertex. The result is a tree spanning all vertices in $\{G_0, G_1, \ldots G_n\}$. Renumbering was accomplished by visiting the vertices in a depth-first traversal of the tree. A vertex of graph G_j was assigned an index of i if it was the i^{th} vertex of G_j to be visited in the depth-first traversal.

Figure 2 illustrates a sample tree for a graph with four levels. In this figure, solid lines represent parent/child relationships in the tree and dotted lines indicate

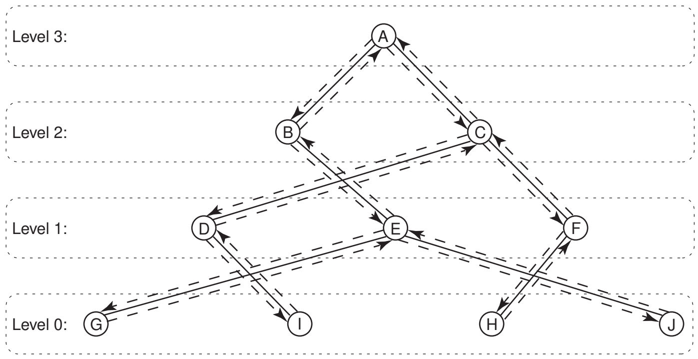

Figure 2: Storage of data structures before reordering

the path of the depth-first traversal during reordering. A vertex's memory address relative to other vertices in the same graph is indicated by its horizontal position. If a vertex is to the left of another vertex in the same level then it is stored at a lower memory address.

The result of rearranging the storage of vertices in Figure 2 using MGR is illustrated in Figure 3. The depth-first traversal visited vertex B and its descendants E, G, and J before vertex C and its descendants. As a result B and its descendants appear to the left of C and its descendants reflecting their placement at lower memory addresses.

The pseudocode for MGR follows:

Algorithm 4 – Multilevel Graph Reorder

Given $\{G_0, G_1, \ldots G_n\}$, find permutation of graph Laplacian L_0 which concentrates non-zero entries near the diagonal.

1. Extend $\{G_0, G_1, \ldots G_n\}$ with additional coarse graphs until a graph with a single vertex is obtained.
2. Using mapping of vertices in G_j to vertices of G_{j+1} construct tree such that vertices of G_j are children of vertices of G_{j+1}.
3. Traverse tree in depth-first manner. Assign new index to each unvisited node. An index i is assigned to a vertex of graph G_j if it is the i^{th} vertex of G_j to be visited during the depth-first traversal of the tree.
4. Update data structures to reflect new order:

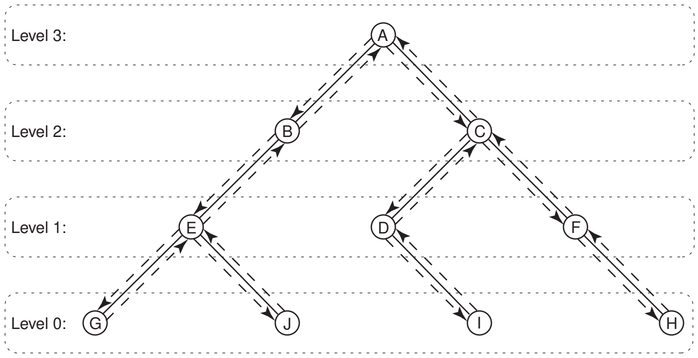

Figure 3: Storage of data structures after reordering

- Update contents of data structures to reflect new indices assigned to vertices.
- Reorder data structures in memory such that data structures corresponding to vertex v_0 have lowest memory addresses followed by data structures for v_1 and so on.
- Sort the edge list of vertices.

For the complexity analysis of the above algorithms, we assume k_0 to be the number of iterations at the finest level, $|V|$ the number of vertices, and $|E|$ the number of edges. In analyzing the time complexity of MDA, it is useful to break down the overall time into the time spent performing calculations at each level. The operations performed by the preconditioner with the greatest time complexity are relaxations. The time complexity of a single relaxation at level i is proportional to $|E_i|$, the number of edges of G_i. During the coarsening process, intermediate graphs were removed to ensure a minimum ratio of the number of vertices at level i to the number of vertices at $i+1$. Let r be the value of this ratio and require $r > 1$ ($r = 4$ for the experiments in this paper). Assume that a similar ratio holds for the number of edges. The time for a relaxation, interpolation or restriction at level i is bounded by $C\,|E_i|$. Since level i is part of $i+1$ V-cycles, the amount of time spent performing operations on level i is bounded by $C'(i+1)|E_i|$ for some constant C'. Thus the total time for the operations at all levels is

$$\text{Total time} \leq C' \sum_{i=0}^{\#\text{ levels}} (i+1)|E_i|.$$

Since $|E_{i+1}| \leq (1/r)|E_i|$, the total time is bounded by

$$C' \sum_{i=0}^{\#\text{ levels}} \frac{1}{r^i}(i+1)|E_0| \leq C' \sum_{i=0}^{\infty} \frac{1}{r^i}(i+1)|E_0| = C' \frac{r^2}{(r-1)^2}|E_0|.$$

Provided $r > 1$ is bounded away from one, the total time is linear in $|E_0| = |E|$. The space complexity of MDA is $O(|V|)$.

For the complexity analysis of NDA, we assume the number iterations at level i is $O(k_0)$ where k_0 is the number of iterations required by NDA at level 0, the finest level. Matrix-vector multiplications each require $O(|E_i|)$ time and orthogonalizing a vector against m orthonormal vectors requires $O(m|V_i|)$ time. An $O(k^3)$ QR method was used to find the eigenpairs of the dense matrix in the Rayleigh-Ritz process. Since, at a given level, this was done once for $k = 1, 2, \ldots O(k_0)$, solving the eigenpairs for the projection into the subspace involved $O(k_0^4)$ work. At level i, $O(k_0)$ iterations of the Davidson algorithm are performed requiring a total of $O(k_0|E_i|)$, $O(k_0|E_i|)$, $O(k_0^2|V_i|)$, and $O(k_0^4)$ time for multilevel preconditioning, matrix-vector multiplication, orthogonalization, and Rayleigh-Ritz' eigenproblem, respectively. The total time complexity of NDA at level i is $O(k_0|E_i| + k_0^2|V_i| + k_0^4)$. Since NDA is run once for each level, the total time required by NDA for all levels can be found by summing over i to give $O(k_0 \sum_i |E_i| + k_0^2 \sum_i |V_i| + k_0^4)$. Since $|V_i|$ and $|V_{i+1}|$ satisfy a minimum ratio r, the sum $\sum_i |V_i|$ is linear in $|V|$, i.e. $\sum_i |V_i| = O(|V|)$. Assuming that the edges also satisfy a similar ratio gives $\sum_i |E_i| = O(|E|)$. The overall time complexity of NDA becomes $O(k_0|E| + k_0^2|V| + k_0^4)$.

NDA computes the Fiedler vector at level $i+1$ before performing any significant computation at level i. The space required for the recursion can be reclaimed before allocating space for the current level. Whereas the regular Davidson algorithm starts with a single basis vector, NDA allows for as many as l. Typically $O(l|V|)$ additional storage is required to accommodate the additional $l-1$ basis vectors. NDA can be run in a mode in which all basis vectors at level $i+1$ are interpolated onto level i and used to seed the Davidson algorithm. Not only are all basis vectors from level $i+1$ interpolated onto level i, but every iteration at level i adds an additional basis vector. The number of bases for level i is the *cumulative* total created for that level and all coarser levels. Assuming convergence after $O(k_0)$ iterations, each of the $O(\log |V|)$ levels will contribute $O(k_0)$ basis vectors, each of length $|V|$. In this case $O(k_0|V|\log|V|)$ storage will be required.

Multilevel Graph Reordering has two phases. The first phase relabels the vertices of a graph and the second phase reorders the data structures based on that relabeling. The coarsening process gives rise to a tree involving vertices of the graphs $G_0, G_1, \ldots, G_n$. The parent-child relationship of the tree reflects which vertices of G_{i-1} were combined to form a vertex in G_i. The tree is a by-product of the coarsening process and does not increase the complexity of Multilevel Graph Reordering. Relabeling the vertices is accomplished via a depth-first traversal of this tree. Relabeling entails a constant amount of work at each vertex visited in the traversal, therefore the time complexity of relabeling is that of the depth-first traversal, $O(|E|+|V|)$, or $O(|E|)$ under the assumption that the graph is connected because $|E|$ will be at least $|V|-1$.

The reordering phase of Multilevel Graph Reordering was implemented as a three step process - a copy of all vertex and edge data structures were made, the data was copied back into the original memory in the new order, and then the edges for each vertex were sorted into increasing order. Moving the data structures to and from temporary buffers entailed a constant amount of work for each vertex and edge as did the update of the contents to reflect the relabeling. Sorting the edges has the potential of being more time consuming. In the general case, sorting E_{v_i}, the set of edges for vertex v_i has $O(|E_{v_i}|\log|E_{v_i}|)$ time complexity, making the reordering phase $O(|V|^2\log|V|)$ in the worst case where $|E| = O(|V|^2)$. In practice, the graphs associated with large, sparse matrices typically have bounded degree. For *bounded degree graphs*, the time required to sort the edges for an individual vertex is $O(1)$ and the reordering phase is $O(|E|+|V|)$. Under the assumptions that a graph is connected and has bounded degree, the overall time complexity of Multilevel Graph Reordering is $O(|E|)$. Again, in the worst case, Multilevel Graph Reordering is $O(|V|^2\log|V|)$.

The implementation of Multilevel Graph Reordering used in this paper used a constant amount of auxiliary storage for each vertex and edge resulting in a space complexity of $O(|E|+|V|)$. Doing so did not increase the overall space complexity of the program because $O(|E|+|V|)$ storage was used to store the graph data structures in memory anyway. The amount of auxiliary storage can be reduced by reordering the data structures in place but $O(|V|)$ storage is still required to store relabeling information.

4. Numerical Experiments

The effectiveness of PDA, NDA, and MGR were gauged by their performance partitioning a suite of test cases. We used the Metis software of Kumar and Karypis, as a framework for our new algorithms. The Metis software provided useful data structures and supporting functionality such as graph coarsening, file IO and timing facilities. For more information about our codes, please refer to [18, 19].

The tests were conducted on an SGI Origin 2000 with 8 gigabytes of RAM and 250 MHz R10000 CPUs, each with 4 megabytes of secondary cache. The tests were run in a dedicated environment.

The Fiedler vector was found for the finest graph in all runs. A vector was deemed to have converged to the Fiedler vector when the l_2 norm of the residual was less than 10^{-5}. Graphs were coarsened using the Heavy Edge Matching strategy [21]. For all graphs, G_n (the coarsest graph) had at most 100 vertices. A 4:1 ratio was imposed on the number of vertices in G_j and G_{j+1} by removing intermediate graphs as necessary.

The suite of test cases were obtained by extracting graphs from large, symmetric matrices from scientific applications [9]. The matrices used in the experiments range in size from 12,000 by 12,000 to more than 200,000 by 200,000. The number of edges ranges from approximately 300,000 to more than 11 million. The test cases

Table 1: Performance of Lanczos and MDA Algorithms

Graph	Vertices	Edges	Lanczos Time	Lanczos Iterations	Davidson Time	Davidson Iterations
3dtube	45330	3168288	89.8	236	46.8	20
bcsstk31	35586	1145826	64.2	437	17.9	18
bcsstk32	44609	1970092	118.5	482	26.6	17
bcsstk35	30237	1419926	65.5	377	21.4	20
cfd1	70656	1757708	76.4	313	29.0	16
cfd2	123440	2964458	153.4	367	62.3	20
ct20stif	52329	2646134	126.8	390	26.1	12
finan512	74752	522240	71.2	683	23.0	19
gearbox	107624	6500976	501.4	636	453.3	87
nasasrb	54870	2622454	95.9	296	40.0	20
pwt	36463	289588	30.1	693	8.6	19
pwtk	217918	11416506	1139.5	807	378.1	42

come from diverse problem domains ranging from financial portfolio optimization methods to stiffness matrices for automobile components.

The performance of the new algorithms were explored through a series of four numerical experiments. These experiments assess the new algorithms when used individually (Sections 4.2,4.3 and 4.4) and in combination (Section 4.1).

4.1. Collective Performance of MDA vs. Lanczos

The first experiment tests the overall effectiveness of MDA (PDA+NDA+MGR) against the widely used Lanczos algorithm. Numerical data from this experiment are shown in Table 1. The table lists the time, in seconds, and the number of iterations required to obtain the Fiedler vector. The time required by the new algorithms is considerably less than that of Lanczos for all graphs in the test suite.

Figure 4 shows the ratio of the time required by the modified Davidson algorithm to the time required by the Lanczos algorithm. All values are less than 1.0 indicating that MDA was faster for all of these graphs. Most values are between 0.3 to 0.5 indicating that MDA is usually 2 to 3 times as fast. MDA was as much as 5 times as fast for the graphs **bcsstk32** and **ct20stiff**. Only the **gearbox** graph had a ratio significantly more than 1:2.

Figure 5 shows the l_2 norm of the residual versus iteration number for MDA. The plot has one data point for each iteration for each graph. The plot shows that all but two of the graphs converged at a similar rate. The norm of the residual of these graphs decreased by roughly one order of magnitude every six iterations. The graphs **pwtk** and **gearbox** showed anomalously slow convergence. **Pwtk** started off converging at a rate comparable to that of the other graphs but slowed considerably after ten iterations. **Gearbox** exhibited slow convergence from the start. The residual of **gearbox** decreased roughly one order of magnitude for each

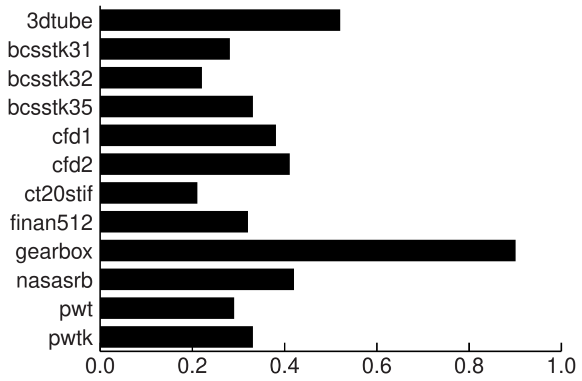

Figure 4: Ratio of MDA to Lanczos execution time

twenty iterations.

The Lanczos and MDA algorithms are two methods for finding the same vector. It was not surprising that the number of edges cut for eight out of twelve graphs was identical whether Lanczos or MDA was used to find the Fiedler vector. The difference in edge cut was one for **bcsstk32**. Lanczos produced a slightly better edge cut for **cfd1** (9969 vs. 10046 edges). MDA produced a slightly better edge cut for **gearbox** (29710 vs. 29767 edges). MDA produced a significantly better cut for **ct20stiff** (17174 vs. 18287 edges). When the difference in edge cut is small (less than 10), it is likely that the partitions are almost identical except for one or two vertices assigned to different partitions. This can be caused by small differences in the individual entries of the Fiedler vector. When the difference is large, the difference in edge cut can be attributed to convergence to nearby but distinct eigenpairs.

4.2. Componentwise Performance of MDA: Multilevel Preconditioner (PDA)

The effectiveness of the multigrid preconditioner is explored by the this experiment. The performance of the multilevel preconditioner of PDA was assessed by comparing its performance against the diagonal preconditioner. The numerical performance of the diagonal preconditioner is not as good as the multilevel preconditioner but it is very fast.

Table 2 presents the execution time, in seconds, and number of iterations. Using the multilevel preconditioner required both fewer iterations and less time to compute the Fiedler vector. For all but three graphs, using the diagonal preconditioner took four or more times as long as the multilevel preconditioner. The multilevel preconditioner gave a speedup of more than ten for some of the graphs. The reduction in the number of iterations was sufficient to compensate for the additional expense of the multilevel preconditioner.

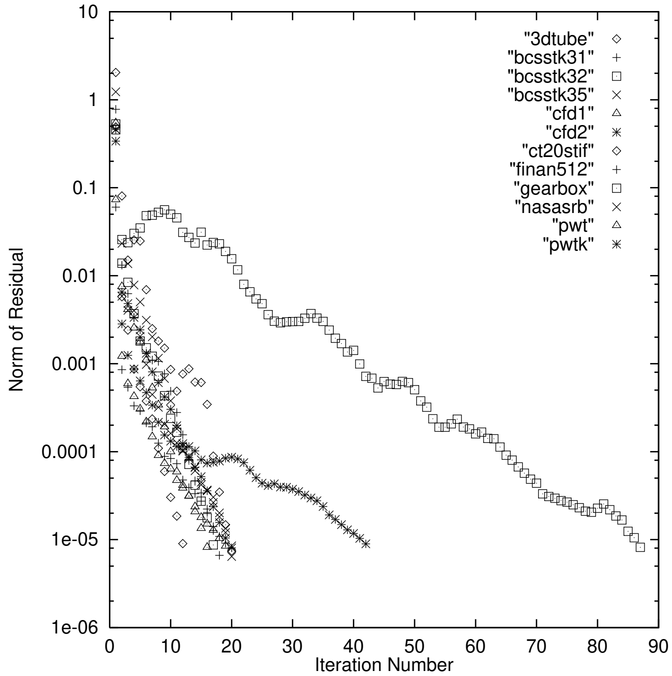

Figure 5: l_2 norm of residual as a function of iteration number.

Table 2: Performance of Multilevel and Diagonal Preconditioners

Graph	Multilevel Time	Multilevel Iterations	Diagonal Time	Diagonal Iterations
3dtube	46.8	20	93.4	98
bcsstk31	17.9	18	100.4	138
bcsstk32	26.6	17	129.3	134
bcsstk35	21.4	20	127.7	160
cfd1	28.9	16	160.1	132
cfd2	62.6	20	323.1	146
ct20stif	26.1	12	59.8	65
finan512	23.0	19	327.5	149
gearbox	453.3	87	3010.0	398
nasasrb	39.9	20	77.7	86
pwt	8.8	19	196.6	203
pwtk	378.9	42	4644.0	362

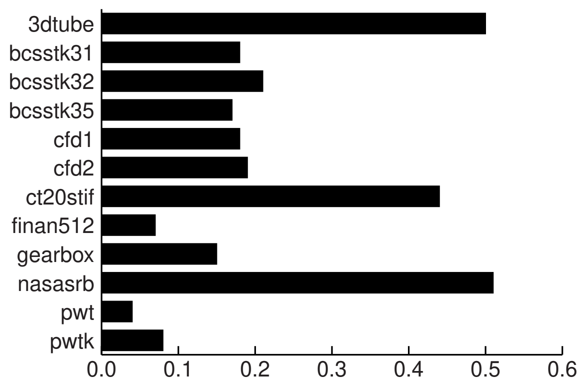

Figure 6: Ratio of multigrid to diagonal preconditioner execution times

Both the diagonal and multilevel preconditioners require approximately constant time per iteration but the time and space requirements of other parts of the Davidson algorithm increase with the number of iterations. For this reason, among preconditioners which perform a constant amount of work per iteration, there is an incentive to trade an increase in the complexity of the preconditioner for a reduction in the number of iterations.

4.3. Componentwise Performance of MDA: Nested Davidson Algorithm (NDA)

The Nested Davidson Algorithm tries to speed the computation of the Fiedler vector of G_0 by starting with a favorable initial subspace. Ideally, the component of the Fiedler vector perpendicular to the initial subspace is small. This experiment studied how well NDA provides a favorable initial subspace.

The experiment compared the performance as a function of l, the dimension of the initial subspace, i.e. the number of basis vectors obtained from the next coarser graph. The value of l ranged from zero to unlimited. Zero represents the case where NDA is not used and unlimited represents the case where the subspace for G_j is initialized with all basis vectors from graph G_{j+1}.

Table 3 presents data from the second experiment. The elapsed time, in seconds, spent executing Davidson algorithm without NDA appear in the second column. NDA is represented by the remaining six columns. In the case where the initial subspace has just one basis vector, the basis vector is the Fiedler vector of G_{j+1}. The times in the last column reflect the case where all basis vectors for G_{j+1} are interpolated onto G_j to form the basis vectors of the initial subspace. Our results show that NDA indeed decreases execution time. While the improvement is often substantial, the amount of improvement is quite variable. Figure 7 shows the ratio of the times required by NDA divided by the time required without NDA. There is a general overall reduction in time when NDA is used and the improvement increases

Table 3: Performance of Nested Davidson Algorithm

Graph	No Bases Time	1 Bases Time	3 Bases Time	5 Bases Time	10 Bases Time	20 Bases Time	All Bases Time
3dtube	52.5	40.8	43.5	44.0	47.2	47.7	33.8
bcsstk31	32.3	23.9	23.6	22.2	21.4	17.9	15.6
bcsstk32	67.5	52.4	51.6	52.4	33.7	26.6	23.8
bcsstk35	44.2	33.6	31.1	32.5	26.6	21.4	17.6
cfd1	88.0	55.4	51.5	49.4	41.2	29.1	31.6
cfd2	141.5	108.1	107.0	97.9	78.4	62.3	66.7
ct20stif	79.4	38.1	36.9	34.1	30.3	26.1	34.2
finan512	44.7	28.1	28.8	29.5	25.2	25.2	27.6
gearbox	488.3	450.9	444.1	428.9	432.8	453.5	236.9
nasasrb	76.7	55.4	39.8	44.0	36.5	40.0	34.5
pwt	42.6	21.2	14.0	13.2	11.4	8.6	10.2
pwtk	1009.8	673.0	669.3	770.8	615.7	378.9	206.3

with the number of bases. For 20 bases, the average speedup approaches 2.

One interpretation of the general improvement with the dimension of the initial subspace is that the additional dimensions represent degrees of freedom. The additional degrees of freedom give the Rayleigh-Ritz process finer control to emphasize certain directions and suppress others when computing the approximation to the Fiedler vector.

Table 4 shows the inner product of the Fiedler vector with its projection in the initial subspace created by NDA for the various graphs. Since the Fiedler vector has unit length, the closer this value is to 1.0, the less the distance between the Fiedler vector and its projection. The majority of the entries have inner products in excess of 0.99 indicating NDA provides a very good initial subspace for the Rayleigh-Ritz process to search for the vector closest to the desired eigenvector.

4.4. Componentwise Performance of MDA: Multilevel Graph Reordering (MGR)

The last experiment tests the effectiveness of the reordering technique. This experiment measured performance when Multilevel Graph Reordering was used against executions when no reordering or random reordering was used. The performance metric was cache line reusage. Cache line reusage is the average number of references made to a cache line before it is evicted from cache.

Cache line reusage is one of the statistics output by the "perfex" performance monitoring utility of SGI computers. The R10000 chip has two counters which can be programed to count events such as memory accesses, cache misses etc. while executing the program at full speed. Perfex accesses these registers to compute a variety of statistics, including cache line reusage.

Table 5 shows cache performance statistics. The value to the left of the slash is the average number of times a cache line in the primary cache was reused before being evicted and the number to the right is the average number of times a line in

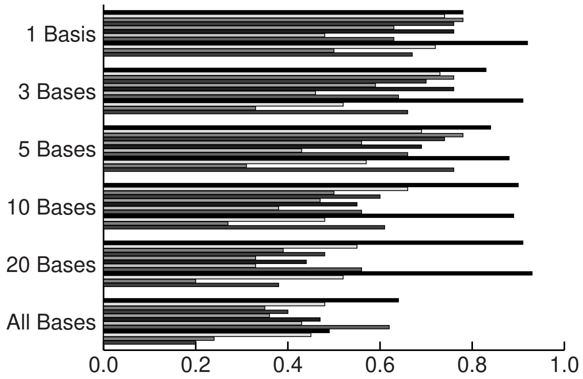

Figure 7: Ratio of nested to non-nested Davidson algorithm execution times

Table 4: Inner Product of Fiedler Vector With Its Projection in Initial Subspace

Graph	Second Finest Graph	Finest Graph
3dtube	0.30263	0.99174
bcsstk31	0.99784	0.99954
bcsstk32	0.99869	0.99982
bcsstk35	0.99799	0.99962
cfd1	0.99918	0.99973
cfd2	0.99933	0.99978
ct20stif	0.99741	0.99961
finan512	0.99861	0.99995
gearbox	0.78122	0.84086
nasasrb	0.99980	0.99993
pwt	0.99966	0.99994
pwtk	0.99970	0.99998

Table 5: Average Primary and Secondary Cache Line Reusage

Graph	Multilevel Graph Reorder (Prim. / Sec.)	No Reorder (Prim. / Sec.)	Random Reorder (Prim. / Sec.)
3dtube	13.9 / 5.3	11.2 / 6.6	9.8 / 7.8
bcsstk31	11.8 / 6.4	9.9 / 6.8	9.4 / 8.3
bcsstk32	12.6 / 5.7	10.9 / 6.2	10.2 / 6.8
bcsstk35	12.4 / 5.7	10.9 / 6.6	9.6 / 7.9
cfd1	10.5 / 6.2	8.5 / 7.7	7.2 / 9.4
cfd2	10.4 / 5.8	7.9 / 7.5	7.1 / 8.0
ct20stif	14.1 / 5.4	10.9 / 6.8	10.1 / 7.5
finan512	7.7 / 9.0	6.1 / 10.5	5.8 / 11.0
gearbox	11.2 / 5.2	9.2 / 6.4	8.0 / 7.0
nasasrb	12.5 / 5.4	11.0 / 6.2	10.1 / 7.1
pwt	8.3 / 10.7	6.4 / 12.9	6.5 / 12.8
pwtk	11.8 / 4.6	9.6 / 5.8	8.7 / 5.5

the second level cache was reused. The data reflect the cache usage of the whole program rather than just the computation of the Fiedler vector.

The data in Table 5 show that MGR had a definite positive impact on improving hit rates for the primary cache. The average number of times a line in the primary cache was reused under MGR was 18% more than if no reordering was done.

5. Summary

Spectral partitioning can be derived by casting the graph partitioning problem as a discrete optimization problem. The new algorithms presented here preserve the solution to the discrete optimization problem by finding the Fiedler vector of the input graph. However, the new algorithms use the multilevel representation of the graph provided by coarsening the graph to speed up the computation of the Fiedler vector.

The Davidson algorithm was used to compute the Fiedler vector. Three new algorithms were developed to speed up the Davidson algorithm. The Preconditioned Davidson Algorithm used the multilevel representation of the graph as the framework for a multilevel preconditioner. The Nested Davidson Algorithm found a favorable initial subspace with which to start the Davidson iterations. Multilevel Graph Reordering used the multilevel representation to reorder the storage of the graph in memory to enhance locality of memory references.

The Preconditioned Davidson Algorithm and Nested Davidson Algorithm proved to be quite effective at reducing the time required to find the Fiedler vector for a range of graphs. Multilevel Graph Reordering met the goal of improving overall cache hit ratios.

Acknowledgements

This research was supported by NSF grant ASC 9528912 and currently by NSF grant DMS 9996089. We thank the referees for advice which has improved the paper presentation. Part of this research was carried out in the Computer Science Department and Supercomputer Center of Texas A&M University.

References

1. J. Atkins, E. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28:297–310, 1998.
2. S. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications*, 2:317–334, 1995.
3. S. Barnard and H. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, 1993. SIAM, SIAM.
4. L. Borges and S. Oliveira. A parallel Davidson-type algorithm for several eigenvalues. *Journal of Computational Physics*, (144):763–770, August 1998.
5. W. L. Briggs. *A Multigrid Tutorial.* Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1987.
6. F. R. K. Chung. *Spectral Graph Theory.* American Mathematical Society, Providence, Rhode Island, 1997.
7. E. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics*, 17:87–94, 1975.
8. E. Davidson. Super-matrix methods. *Computer Physics Communications*, 53:49–60, 1989.
9. T. Davis. University of florida sparse matrix collection. March 1998. http://www.cise.ufl.edu/ davis/sparse/.
10. C. Douglas. Caching in with multigrid: problems in two dimensions. *Parallel Algorithms and Applications*, 9:195–204, 1996.
11. M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305, 1973.
12. M. Fiedler. A property of eigenvectors of non-negative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25:619–632, 1975.
13. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.
14. A. George and A. Pothen. An analysis of spectral envelope reduction via quadratic assignment problems. *SIAM Journal on Matrix Analysis and Applications*, 18:706–732, 1997.
15. G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, Maryland, 2nd edition, 1989.
16. B. Hendrickson and R. Leland. The chaco user's guide, version 2.0. Technical Report SAND95-2344 SAND95-2344, Sandia National Laboratories, Alberquerque, NM 87185-1110, 1994.
17. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computin*, 16:452–

469, 1995.

18. M. Holzrichter and S. Oliveira. New graph partitioning algorithms. December 1998. The University of Iowa TR-120.

19. M. W. Holzrichter. *New Spectral Partitionings Algorithms*. PhD thesis, Texas A&M University, 1998.

20. G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. Technical report tr 95-037, Department of Computer Science, University of Minnesota, Minneapolis Minnesota, 1995.

21. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

22. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, February 1970.

23. S. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1989.

24. S. Oliveira. A convergence proof of an iterative subspace method for eigenvalues problem. In F. Cucker and M. Shub, editors, *Foundations of Computational Mathematics Selected Papers*, pages 316–325. Springer, January 1997.

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

26. A. Pothen, H. D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990. Sparse matrices (Gleneden Beach, OR, 1989).

27. A. Pothen and S. Ye. Enhancing the cache performance of irregualr computations. In *Book of Abstracts of the 1998 SIAM Annual Conference*, Norfolk, Virginia, 1998. SIAM.

28. Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992.

29. H. D. Simon and S. H. Teng. How good is recursive bisection. *SIAM Journal on Scientific Computing*, 18(5):1436–1445, July 1997.

30. D. Spielman and S. H. Teng. Spectral partitioning works: planar graphs and finite element meshes. In *37th Annual Symposium Foundations of Computer Science*, Burlington, Vermont, October 1996. IEEE, IEEE Press.

31. S. Toledo. Locality of reference in lu decomposition with partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 18:1065–1081, 1997.

32. J. Y. Zien, M. D. Schlag, and P. K. Chan. Multi-level spectral hypergraph partitioning with arbitrary vertex sizes. In *Proceedings of International Conference on Computer Aided Design*. IEEE Press, 1994.