INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 40, 1025–1051 (1997)

# EVALUATION OF AUTOMATIC DOMAIN PARTITIONING ALGORITHMS FOR PARALLEL FINITE ELEMENT ANALYSIS

SHANG-HSIEN HSIEH*

*Department of Civil Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.*

GLAUCIO H. PAULINO† AND JOHN F. ABEL‡

*School of Civil and Environmental Engineering, Hollister Hall, Cornell University, Ithaca, NY 14853-3501, U.S.A.*

## SUMMARY

This paper studies and compares the domain partitioning algorithms presented by Farhat,[1] Al-Nasra and Nguyen,[2] Malone,[3] and Simon[4]/Hsieh *et al.*[5,6] for load balancing in parallel finite element analysis. Both the strengths and weaknesses of these algorithms are discussed. Some possible improvements to the partitioning algorithms are also suggested and studied. A new approach for evaluating domain partitioning algorithms is described. Direct numerical comparisons among the considered partitioning algorithms are then conducted using this suggested approach with both regular and irregular finite element meshes of different order and dimensionality. The test problems used in the comparative studies along with the results obtained provide a set of benchmark examples for other researchers to evaluate both new and existing partitioning algorithms. In addition, interactive graphics tools used in this work to facilitate the evaluation and comparative studies are presented.

KEY WORDS: algorithms; domain partitioning; load balancing; mesh partitioning; parallel finite element analysis; partitioning algorithms

## 1. INTRODUCTION

A number of domain partitioning techniques have been proposed in recent years to effect load balancing in parallel finite element analysis. The domain partitioning techniques decompose the domain of a finite element mesh into a number of subdomains which are distributed among processors for computation. Well-balanced distribution of computations among subdomains (or processors) and minimization of interprocess communication are sought so that significant speed-up can be obtained in the parallel analysis. The domain partitioning techniques are also important for domain decomposition solution methods in computational mechanics. For a thorough survey of the general field of domain decomposition methods, see Reference 7.

* Lecturer of Computer-Aided Engineering
† Graduate Research Assistant
‡ Professor of Structural Engineering

CCC 0029–5981/97/061025–27

*Received 17 May 1994*
*Revised 17 July 1996*

Because domain partitioning is an NP-complete problem,[8] optimum solutions are practically intractable. Therefore, satisfactory near-optimal solutions are always sought using efficient heuristic algorithms. An optimization approach known as Simulated Annealing (SA[§]) has been proposed by several researchers[9,10] to solve the partitioning problem. Although this approach can give almost optimal results, it is usually expensive and time-consuming, especially for large-scale problems. This paper studies and compares several alternative partitioning approaches proposed in the literature which seek satisfactory near-optimal solutions at a relatively low computational cost.

Because the finite element meshes used in solving engineering problems may be large and generic (irregular shapes, including multiply connected and/or branched domains), manual partitioning may be difficult even with the help of interactive graphics tools. Simon[4] has shown that visual perception alone may be inadequate for the task of partitioning large three-dimensional meshes. Therefore, automatic algorithms are ideally required. On the other hand, none of the existing automatic partitioning algorithms guarantees optimal solutions or produces partitions that generally meet all the needs of various numerical solution strategies. Therefore, Farhat and Lesoinne[11] proposed a 'multiple choice solution' which obtains partitions from several automatic partitioning algorithms and then selects the best for the parallel solution algorithm considered. In addition to the 'multiple choice solution' approach, Hsieh[5] used interactive graphics tools to allow for manually improving results obtained from automatic partitioning. In this paper, the emphasis is on automatic heuristic partitioning algorithms.

### *1.1. Classification of algorithms*

The techniques considered here are *static load balancing* techniques because (a) they assume *a priori* knowledge of the static characteristics of the computations and the system to distribute the computations among processors and (b) the distribution is done only once and before the actual analysis starts (i.e. in the preprocessing phase). This is opposed to the *dynamic load balancing* (sometimes referred to as *adaptive load balancing*) techniques which utilize short-term knowledge of current state information of the computations and the system to distribute dynamically the computations among processors during the analysis. The computations originally assigned to a processor may be migrated (or redistributed) to other processors at any time during the analysis to improve load balance. Dynamic load balancing may be important in problems involving material non-linearity, self-adaptive mesh refinement, crack propagation, etc.

For the sake of discussion in this paper, most of the existing automatic heuristic domain partitioning algorithms may be classified in the following ways.

*Topology-based vs. geometry-based*: The *topology-based* algorithms partition a mesh using topological information of the mesh or its associated graph (for background material in graph theory, see the excellent book by Harary[12]). For example, the GReedy (GR) algorithm,[1] the Reduced Bandwidth Decomposition (RBD) algorithm,[3,13] the Recursive Graph Bisection (RGB) algorithm,[4] and the Recursive Spectral algorithms[4,6,14,15] belong to this category. On the other hand, the *geometry-based* algorithms partition a mesh using geometrical information. Examples are the Recursive Co-ordinate Bisection (RCB) algorithm[4] and the inertial algorithms.[11,16] In addition, there are algorithms which use both topological and geometrical information for partitioning. For example, Al-Nasra and Nguyen's Partitioning (ANP) algorithm,[2] the

[§] Abbreviations of the partitioning algorithms used in this paper are summarized in the appendix

'combination algorithm' of Rodriguez and Sun,[17] the 'automated substructuring' approach of Padovan and Kwang,[18] and the algorithm by Miller *et al.*[19] belong to this category.

*Spectral vs. non-spectral*: Spectral methods, based on algebraic properties of a certain graph associated with the finite element mesh, have been proposed to solve the partitioning problem (see, for example, References 20 and 21). Algorithms based on these methods can be classified as *spectral* algorithms, while the others are *non-spectral* ones. The spectral methods use *global* properties of the graph associated with the mesh to perform the partitioning, i.e. a spectral analysis is conducted to compute separators based on eigenvector components of the graph. This is opposed to most of the non-spectral algorithms which, in general, use only *local* information in the graph, such as the neighbouring information of a vertex. Several spectral partitioning algorithms have been presented in the literature. For example, Simon[4] proposed the Recursive Spectral Bisection (RSB) algorithm for hypercube architectures, which consist of $2^d$ processors ($d$ is the dimension of the hypercube). Hendrickson and Leland[14,15] extended the spectral bisection through the use of multiple eigenvectors to allow for partitioning of a domain into 4 or 8 subdomains at each stage of a recursive decomposition. They proposed the Recursive Spectral Quadrisection (RSQ) algorithm for $4^k$ partitions ($k$ is a positive integer number) and the Recursive Spectral Octasection (RSO) for $8^k$ partitions. However, all these algorithms (RSB, RSQ, and RSO) have been specifically developed for the hypercube or mesh (machine) architectures. To generalize the RSB algorithms for an arbitrary number of processors, Hsieh[5] and Hsieh *et al.*[6] presented the Recursive Spectral Sequential-cut (RSS) and the Recursive Spectral Two-way (RST) algorithms. The RSB, RSQ, RSO, RSS, and RST algorithms are based on spectral properties of the Laplacian matrix[22,23] of a graph associated with the mesh.

*Recursive vs. non-recursive* (*sequential*): The partitioning algorithms are either recursive in nature or not. For example, the GR, RCB, RGB, RSB, and RST algorithms are *recursive*, while the RBD and PI algorithms are *non-recursive*. For some partitioning techniques, there may exist both recursive and sequential versions. For example, the sequential versions of the RCB, RGB, and RSB algorithms have been studied and compared with their recursive counterparts by Venkatakrishnan *et al.*[24] It is found that the sequential versions tend to produce stripwise partitions with longer boundaries but fewer neighbours, while the recursive ones often generate domainwise partitions with smaller aspect ratios[¶], shorter boundaries, but more neighbours. Similar findings have also been reported by Farhat and Simon[25] between the Reverse Cuthill–McKee-based partitioning (RCM[‖]) algorithm, and the Recursive RCM (RRCM) algorithm.

### *1.2. Comparative studies*

Several comparative studies among some automatic partitioning algorithms have been reported. Fox[26] reviewed five automatic load balancing and decomposition methods for the hypercube computer: (1) scattered decomposition, (2) self-scheduling, (3) Orthogonal Recursive Bisection (ORB), (4) SA and (5) Neural Networks. According to Fox,[26] the first two algorithms are easy to use and work well for a broad class of problems, while the remaining three algorithms represent more powerful techniques that are generally applicable but require a more elaborate environment on the hypercube computer. Simon[4] compared the RCB, RGB, and RSB algorithms

[¶] The subdomain aspect ratio is defined here as $H_{max}/H_{min}$, where $H_{min}$ and $H_{max}$ are the smallest and largest distances between two boundary nodes of the subdomain, respectively

[‖] Note that both the *RCM reordering* and the *RCM based partitioning* algorithms have the same abbreviation

and showed the superiority of the RSB algorithm over both the RCB and the RGB algorithms. Williams[27] evaluated the performance of three partitioning algorithms for dynamic load balancing using a 16-processor NCUBE machine. He compared the SA, ORB, and Eigenvector Recursive Bisection (ERB—essentially the same as the RSB algorithm) algorithms. He concluded that the ERB algorithm seems to be a good compromise between the other two algorithms. In numerical comparative studies using finite element meshes of different types, Hsieh[5] demonstrated that the RST algorithm is a good generalization of the RSB algorithm. He also showed that the RST algorithm, when used with the communication graph approach, produces the best results among the GR, ANP, and RBD algorithms for most cases studied. Farhat and Lesoinne[11, 16] compared the GR, minimum bandwidth, inertial, and recursive inertial algorithms, and indicated that the GR algorithm performs the best in all the reported examples. Very recently, Farhat and Simon[25] compared the GR, RCM, RRCM, Principal Inertial (PI), Recursive Principal Inertial (RPI), RGB, RSB, and 1D Topology Frontal (1DTF) algorithms. They have reported that, for the three test problems studied, 'the GR and RSB algorithms outperform all of the other partitioning algorithms in reducing the interface size' (Reference 25, p. 19).

The comparative studies mentioned above have provided some useful information for engineers to select appropriate partitioning algorithms that meet the needs of a specific parallel solution strategy. However, these studies have not fully addressed the sensitivity of partitioning algorithms with respect to the following:

1. The granularity of partitions (i.e. coarse-grained vs. fine-grained partitioning).
2. Slight variations in the number of partitions required.
3. Finite element meshes of similar shapes but of varying mesh density and element interpolation order.
4. Nodal and/or element numbering of a finite element mesh.

Recently, Hsieh and Paulino[28] have presented a critical assessment of partitioning algorithms which addresses the above issues. This paper presents a more comprehensive study which uses both regular and practical (generic or non-structured) finite element meshes to evaluate the above and other properties of domain partitioning algorithms.

In this work, the GR, ANP, RBD, and RST algorithms are selected for comparative studies and demonstration of the proposed evaluation methodology. Whether or not these algorithms are widely used is not the major criterion used for selection. These four algorithms represent four different approaches and have apparent characteristics that are worth investigating. For example, the GR algorithm is a simple topology-based algorithm, while the ANP algorithm uses both topological and geometrical information. The RBD algorithm is based on bandwidth reduction technique and is non-recursive. The RST algorithm is based on spectral methods and is recursive. Moreover, the strengths and weaknesses of these algorithms are also critically studied.

### *1.3. Organization*

The remainder of this paper is organized as follows. Section 2 briefly describes the four partitioning algorithms investigated and their implementation in this work. Both strengths and weaknesses of these algorithms are then discussed in Section 3. Possible improvements to some of these partitioning algorithms are suggested and evaluated. In Section 4, an evaluation methodology for domain partitioning is described. In addition, the interactive graphics program PSAINT (Parallel Structural Analysis Interface) used in this work for implementing and evaluating domain partitioning algorithms is presented. In Section 5, direct numerical comparisons are then conducted using finite element meshes of different regularity, order, numbering, and

dimensionality to study the characteristics and to evaluate the performance of the partitioning algorithms. Finally, several conclusions are drawn and directions for future work are discussed in Section 6.

## 2. THE DOMAIN PARTITIONING ALGORITHMS

Brief descriptions of the four automatic partitioning algorithms investigated are provided below. Their implementation in this work is also described.

*GReedy* (*GR*) *algorithm*:[1] The algorithm is summarized in Table I. The weight of a node is defined as the number of unassigned elements connected to it. The boundary of a subdomain is defined as the subset of its boundary that connects to other subdomains. The present implementation uses the routines provided in the paper by Farhat[1] with some corrections.[29]

*Al-Nasra and Nguyen's partitioning* (*ANP*) *algorithm*:[2] The algorithm is similar to Farhat's algorithm but incorporates the geometrical information of the finite element mesh in the partitioning procedure. A summary of the algorithm is presented in Table II. The present implementation uses the routines provided in the paper by Al-Nasra and Nguyen[2] with slight modifications to improve performance. Moreover, to avoid an endless search for the minimum-weighted node when domain splitting situations occur, Step (3) of the algorithm (see Table II) has been modified as follows:

(3) Locate a node that has a non-zero minimal weight and is not located at the boundaries with other subdomains. If all nodes in the current subdomain are not qualified, perform a search among all unassigned nodes in the mesh.

Table I. GReedy (GR) partitioning algorithm[1]

| |
|---|
| (1) Locate a node that belongs to the boundary of the previously defined subdomains (for the first time, use the whole domain) and has a non-zero minimal weight. |
| (2) Assign unassigned elements that are connected to this node to the current subdomain. Recursively, assign unassigned elements that are adjacent to the elements in the current subdomain to the current subdomain until the number of elements equals to the total number of elements divided by the number of processors. |
| (3) Repeat (1) and (2) until all subdomains are defined. |

Table II. Al-Nasra and Nguyen's Partitioning (ANP) algorithm[2]

| |
|---|
| (1) Assign an initial weight to each node. The initial weight of a node is defined as the number of elements connected to it. |
| (2) Adjust the initial weight of each node based on its geometric location in the mesh such that extra weight is added increasingly along the long direction of the mesh. |
| (3) Locate a node that has a non-zero minimal weight and is not located at the boundaries with other subdomains. |
| (4) Assign unassigned elements that are connected to this node and their associated nodes to the current subdomain, and reduce the weight of the nodes by one. |
| (5) Locate a node with minimum weight and with at least one adjacent unassigned element in the current subdomain. |
| (6) Repeat (4) and (5) until the number of elements equals to the total number of elements divided by the number of processors. |
| (7) Repeat (3)–(6) until all subdomains are defined. |

Table III. Reduced bandwidth decomposition (RBD) algorithm[3]

| |
|---|
| (1) Reduce bandwidth of the matrix representing the nodal connectivities of the finite element mesh. |
| (2) Reorder elements in ascending sequence of their lowest numbered nodes. |
| (3) To each processor, assign the elements in order until the number of elements equals to the total number of elements divided by the number of processors. |

Table IV. Recursive spectral two-way (RST) algorithm[5,6]

| |
|---|
| (1) Construct the dual (DG), communication (CG), or node graph (NG) associated with the finite element mesh. |
| (2) Compute the second eigenvector of the Laplacian matrix (called the *Fiedler vector* by Simon[4]) of the graph using, for example, the Lanczos algorithm. |
| (3) Sort vertices of the graph according to the value of their associated components in the Fiedler vector. |
| (4) Compute the following integers: $p1 = np/2$ (discard remainder), $p2 = np - p1$, and $n = \sum_{k=1}^{p1} m_{l(k)}$<br>Assign $n$ vertices and the list of the first $p1$ components in $l(k)$ to one subdomain, and set $np = p1$ for this subdomain.<br>Assign the remaining vertices and the list of the remaining components in $l(k)$ to the other subdomain, and set $np = p2$ for this subdomain. |
| (5) Repeat recursively for each subdomain with $np > 1$. |

*Reduced bandwidth decomposition* (*RBD*) *algorithm*:[3] The algorithm is summarized in Table III. In Step (1), it is stated by Malone[3] that a modified version of the Collins algorithm[30] has been used. The present implementation uses a modified version of the Collins algorithm developed by the writers.

*Recursive spectral two-way* (*RST*) *algorithm*:[5,6] This algorithm generalizes the RSB algorithm[4] for an arbitrary number of partitions. Instead of using a bisection approach, the RST algorithm uses a two-way partitioning approach which partitions the graph into two parts not necessarily equal in size. The algorithm is given in Table IV. Use of this algorithm in association with the Dual Graph (DG), Communication Graph (CG), and Node Graph (NG) representation of a finite element mesh has been discussed in detail by Hsieh[5] and Hsieh *et al.*[6] and is not repeated here. The number of vertices in each subdomain $D_i$ when the partitioning task is completed is denoted by $m_i$. It is computed in advance by sequentially employing the following equation for $i = 1, \ldots, N_p$:

$$m_i = \left[ \frac{N - \sum_{j=1}^{i-1} m_j}{N_p - (i-1)} \right] (+1 \text{ if remainder} \neq 0) \tag{1}$$

in which $N_p$ is the number of available processors and $N$ is the total number of vertices of the whole domain. The number of partitions desired in the intermediate subdomain in each two-way partitioning step is denoted by $n_p$ (initially $n_p = N_p$ for the whole domain). Each intermediate subdomain maintains a list $l$ of subdomain numbers associated with $D_i$ which has been assigned to it (initially the list is $\{1, \ldots, N_p\}$ for the whole domain). The $k$th component of this list is denoted by $l(k)$. As shown in Table IV, the implementation of the RST algorithm requires the computation of the second eigenvector of the Laplacian matrix. The implementation by Hsieh[5] and Hsieh *et al.*[6] used the Lanczos algorithm presented by Simon[4] for this eigenvector

computation. The present implementation is based on the fast multilevel implementation of the RSB algorithm provided by Barnard and Simon.[31]

### 2.1. Element-based vs. node-based partitioning

Different parallel solution methods used in the finite element analysis may require different strategies for domain partitioning. Two types of partitioning strategies may be classified: *element-based partitioning* and *node-based partitioning*. The element-based partitioning focuses on partitioning elements in finite element meshes, while the node-based partitioning focuses on partitioning nodes. For example, in parallel solutions of structural dynamics, implicit solution methods often require element-based partitioning (see, for example, Reference 32), while explicit methods may require either type depending on the parallel implementation (see, for example, References 3 and 33).

In the partitioning algorithms discussed above, only the spectral partitioning algorithms with the node graph approach have directly addressed the node-based partitioning. For those algorithms which address only element-based partitioning, a simple extra step at the end of the algorithms may be performed to assign common boundary nodes of two or more subdomains uniquely to a subdomain so that the algorithms can also be used for node-based partitioning.[34] In this case, however, the results of the node-based partitioning would be highly dependent on those of the element-based partitioning obtained before the extra step is performed. Therefore, in this work the focus is placed on the comparative studies of element-based partitioning among different algorithms.

## 3. EXAMINATION OF THE DOMAIN PARTITIONING ALGORITHMS

The strengths and weaknesses of the partitioning algorithms described in the previous section are discussed below. The particular classification of the algorithms, presented in the introduction of this paper, is employed here. Numerical evaluation and comparative study of the algorithms are presented later, in Section 5.

### 3.1. The GR algorithm

The GR algorithm is topology-based, non-spectral, and recursive. The algorithm is simple because it uses mainly local element adjacency information recursively to perform partitioning (see Table I). The GR algorithm tends to generate domainwise subdomains and appears effective for fine-grained partitioning.

According to Farhat,[1] the GR algorithm is 'independent of both element and nodal point numbering' (Reference 1, p. 582) since only adjacency information of elements is utilized. However, the decomposition obtained from the GR algorithm is actually not independent of nodal numbering. In Step (1) (see Table I), there may be several nodes with the same minimal weight, and the first node encountered with minimal weight is arbitrarily selected. In this case, the initial nodal numbering determines which starting node is to be selected. It has been found that the selection of the node with minimal weight in Step (1), especially the very first one in the algorithm, greatly affects the decomposition results. In some cases, whether domain splitting occurs in the decomposition may depend on the node selection in Step (1). For example, in the transmission tower shown in Figure 1(a), the six vertices enclosed by a circle are nodes with minimal weight for the first step of the GR algorithm. Figure 1(b) shows the partitioning results

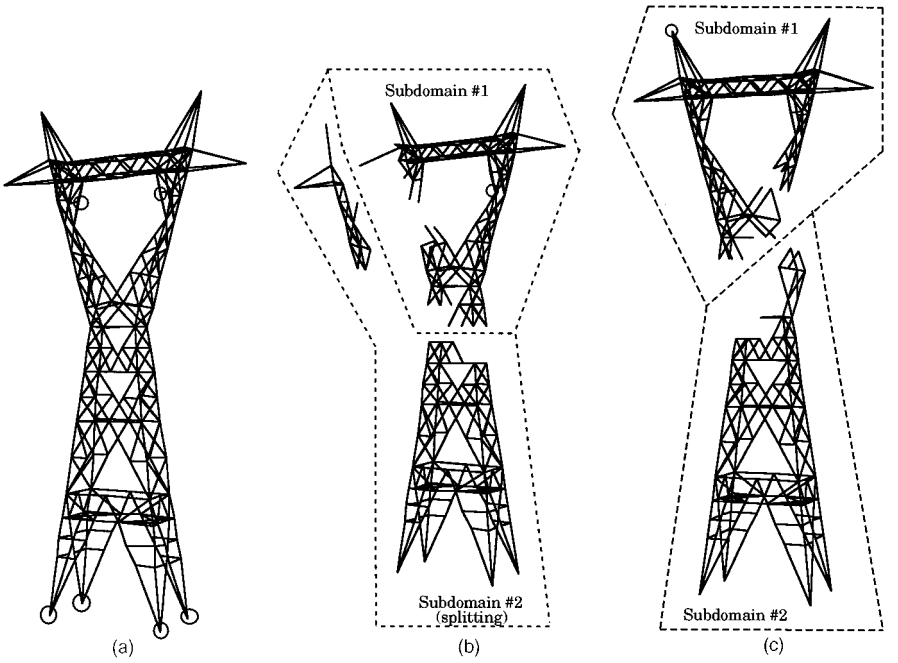


Figure 1. Partitioning of a transmission tower using the GR and PGR algorithms: (a) transmission tower with minimum-weighted vertices circled; (b) partitioning by the GR algorithm starting with the circled minimum-weighted vertex; (c) partitioning by the PGR algorithm starting with the circled pseudoperipheral vertex

for two subdomains with a splitting situation occurring in the second subdomain. In this case, the algorithm uses the upper right minimum-weighted node as the starting point. If a different nodal numbering is used which makes the algorithm start from any of the four support nodes (minimum-weighted nodes), domain splitting will be avoided for this case.

One strategy to attack this problem is to run the GR algorithm trying all minimum-weighted nodes found in Step (1) for the first subdomain, then to compare the partitioning results to select the best. However, this strategy can be computationally expensive because a problem may have many minimum-weighted nodes for the first subdomain.

A preconditioning technique seems to be more efficient in this case. The writers have some evidence that a *peripheral node*** is a good alternative for the starting node in the GR algorithm. Since finding a peripheral node is computationally expensive, the writers have, instead, used the pseudoperipheral node finder presented by George and Liu.[36] However, any other equivalent algorithm can be used for this purpose, e.g. the one presented by Gibbs *et al.*[37] or Paulino *et al.*[35] The implemented Preconditioned GReedy (PGR) algorithm is given in Table V. As an example, consider again the transmission tower of Figure 1(a). Figure 1(c) shows the partitioning results by the PGR algorithm. One can observe that domain splitting does not occur. In addition, the

**A *peripheral node* corresponds to a vertex in the associated graph such that the *eccentricity* $e(v_i) = \max\ d(v_i, v_j)$ $(j = 1, \ldots, n)$ for $i = 1, \ldots, n$, is equal to the *diameter* of the associated graph $\delta = \max\ e(v_i)$ $(i = 1, \ldots, n)$. Here $n$ is the number of vertices of the graph and $d(v_i, v_j)$ is the distance between vertices $v_i$ and $v_j$ (i.e. the topological length of the shortest path connecting $v_i$ and $v_j$). A vertex $v_i$ is peripheral if $e(v_i) = \delta$. For further explanation, see Reference 35.

Table V. Preconditioned GReedy (PGR) partitioning algorithm

| |
|---|
| (0) Find a pseudoperipheral node and use it as a starting node. Go to (2). |
| (1)–(3) Same as the GR algorithm (Table I). |

partitions produced by the PGR algorithm have fewer total interface nodes than the GR algorithm does (16 for the PGR algorithm and 23 for the GR algorithm). However, it should be noted that, in general, the PGR algorithm does not guarantee the absence of domain splitting.

### *3.2. The ANP algorithm*

The ANP algorithm is both topology- and geometry-based, non-spectral, and recursive. The geometrical information of the finite element mesh is incorporated into the algorithm as an attempt to avoid domain splitting in subdomains. However, it will be shown by the writers that the avoidance of domain splitting is still not guaranteed by the ANP algorithm. Moreover, the algorithm tends to generate stripwise subdomains along the 'overall long direction' of the structure.

Al-Nasra and Nguyen[2] concluded in their paper that 'In all the tested problems, domain-splitting phenomena (an undesirable situation) do not occur' (Reference 2, p. 284). However, for some applications tested in the present work, splitting did occur. For example, Figure 2(a) shows an 8-bladed turbine disk modelled by solid finite elements, and Figure 2(b) presents the partitioning results for two subdomains obtained by the ANP algorithm. It can be seen that splitting occurs in the second subdomain. Moreover, when the splitting occurs in a subdomain other than the last one, Step (3) of the original algorithm (see Table II) may go into an endless search for the minimum-weighted node. As already described in the previous section, this problem has been fixed in the present implementation.

In addition, the long and short dimensions defined in the algorithm are in terms of the directions of global axes used to build the finite element model of the structure. This means that different choices of the global axes and different ways of orienting the structure with respect to the

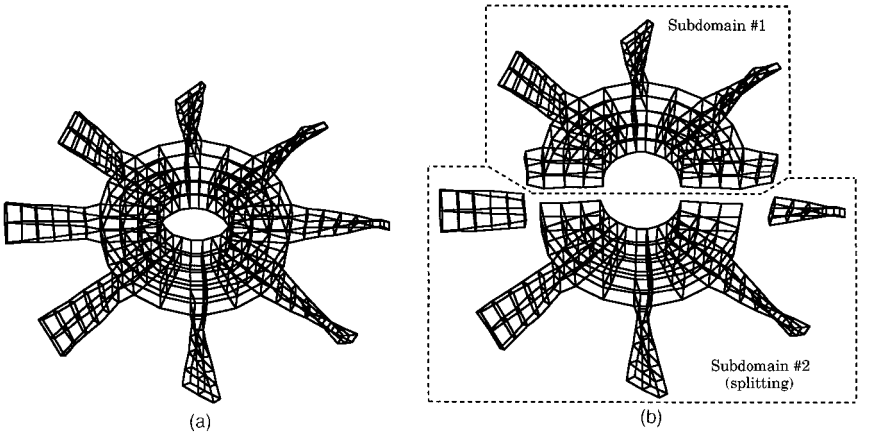


Figure 2. An eight-bladed disk and its partitioning by the ANP algorithm: (a) eight-bladed disk model; (b) partitioning by the ANP algorithm

chosen global axes may result in different partitioning results. Therefore, to obtain good results, caution is needed in preparation of input data for this algorithm.

### 3.3. The RBD algorithm

The RBD algorithm is topology-based, non-spectral, and non-recursive. The algorithm is based on the well-established concept of matrix bandwidth reduction in finite element analysis. As the number of partitions increase, this algorithm tends to generate stripwise subdomains which often result in longer subdomain boundaries but fewer neighbours.

In Step (1) of the RBD algorithm (see Table III), it is stated by Malone[3] that a modified version of the Collins' automatic nodal renumbering algorithm for bandwidth reduction[30] is used. The present implementation uses a modified version of the Collins algorithm developed by the writers. However, Collins' algorithm is not 'effective for most meshes' as stated by Malone (Reference 3, p. 42), who presented only meshes with Constant Strain Triangles (CST). This algorithm was reported to be unsuccessful for meshes with eight-noded quadrilateral elements.[30] Moreover, all the examples presented by Collins[30] are too small when compared to large meshes that demand parallel analysis. Therefore, if a more efficient and effective heuristic algorithm is used in Step (1) to reduce the bandwidth of the system matrix, better partitioning results may be obtained. Possible candidate algorithms are the ones presented by Cuthill and McKee[38] (see Reference 39 for an efficient implementation), Gibbs *et al.*[37] and Puttonen.[40] Some researchers[11,16,25] have unnecessarily used the Reverse Cuthill–Mckee (RCM) algorithm[39] for this purpose. Since the purpose is bandwidth reduction, the plain Cuthill–Mckee (CM) algorithm suffices. The extra step for the reverse numbering is not necessary because both the CM and RCM algorithms give exactly the same bandwidth (see explanations in Reference 41). In the present work, the algorithm presented by Gibbs *et al.*[37] is selected for the purpose of bandwidth reduction. The resulted partitioning algorithm is therefore denoted as the RBD(GPS) algorithm, while the original one presented by Malone[3] is now denoted as the RBD(Col) algorithm.

There is no guarantee in RBD-type algorithms that domain splitting does not occur in the subdomains created. For example, the space station of Figure 3(a) is partitioned by the RBD(Col)

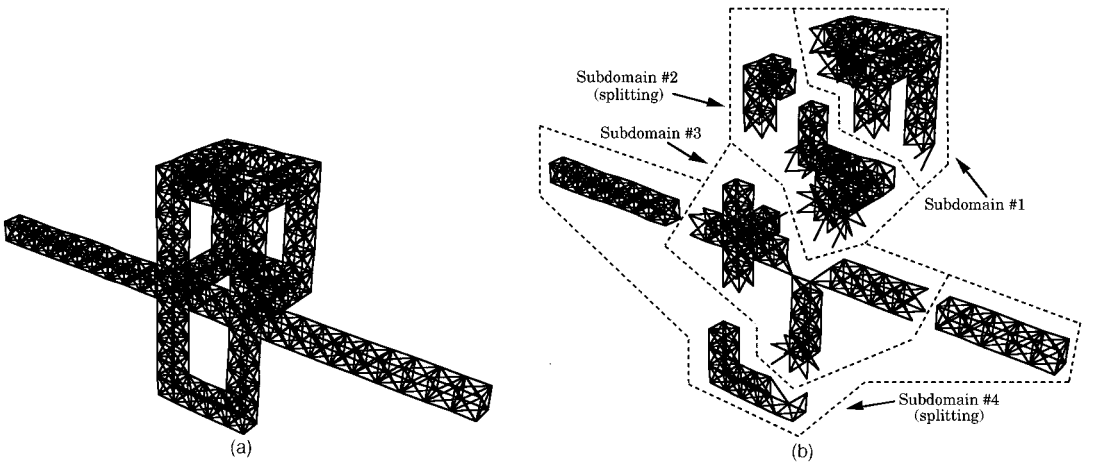


Figure 3. Partitioning of a space station by the RBD(Col) algorithm: (a) space station model; (b) partitioning by the RBD(Col) algorithm

algorithm into four subdomains, as shown in Figure 3(b). It can be seen that splitting occurs in subdomains 2 and 4.

### *3.4. The RST algorithm*

The RST algorithm is topology-based, spectral, and recursive. Being a spectral method, the RST algorithm has the advantage over most non-spectral algorithms of using global information about the associated graph in the partitioning process. However, because the finite element meshes of engineering structures may be large and generic, the eigensolutions required by the spectral analysis in the recursive loop of the algorithm (see Table IV) can be difficult to treat and computationally expensive.

To alleviate the problems mentioned above, the fast multilevel implementation by Barnard and Simon[31] is used in this work for the eigensolution required by the RST algorithm. They have reported that for large problems, their fast multilevel implementation achieves about an order-of-magnitude improvement in run time over the previous single-level implementation by Simon.[4] In the present work, the RST algorithm generalizes the multilevel implementation of the RSB algorithm for an arbitrary number of partitions (see Table IV).

The multilevel technique to find the Fiedler vector of a graph adds three additional steps to the single-level technique: contraction, interpolation, and refinement. These steps are briefly summarized below. For more details, see Reference 31.

*Contraction*: Construct a series of smaller graphs which in some sense retain the global structure of the original (large) graph.

*Interpolation*: Given a Fiedler vector of a contracted graph, interpolate this vector to the next larger graph in such a way that it provides a good approximation to the next Fiedler vector.

*Refinement*: Given an approximate Fiedler vector for a graph, compute a more accurate vector efficiently. A combination of Rayleigh Quotient Iteration (RQI)[42] and the SYMMLQ algorithm[43] (for solution of indefinite systems) are used to refine the coarse grid approximate of the eigenvector being sought.

There are several parameters involved in the multilevel implementation that may affect the performance of multilevel recursive spectral algorithms, such as the present version of the RST algorithm. These parameters are listed below and their default values used in this work are noted within the parentheses.

(a) Maximum Lanczos iteration number (400).
(b) Convergence tolerance for the Lanczos solver (0·0001).
(c) Minimum number of equations, which determines the size of the smallest mesh in the contraction step (750).
(d) Maximum number of vertices for Kernighan–Lin algorithm[44] used to improve the partition after each bisection (300).
(e) Convergence tolerance for the SYMMLQ solver (0·01).
(f) Convergence tolerance for the SYMMLQ version of RQI (0·05).
(g) Maximum number of iterations for the SYMMLQ version of RQI (100).

The writers' experience shows that an adequate tuning of some of these parameters (especially the minimum number of equations for the contraction step) is, in general, necessary for the success of multilevel partitioning algorithms. In addition, the partitioning results are generally dependent on the accuracy of the eigensolutions although only modest accuracy is required for obtaining reasonably satisfactory results. Furthermore, the RST algorithm, as implemented in the present work, does not guarantee that splitting problems do not occur.

## 4. EVALUATION METHOD FOR DOMAIN PARTITIONING

This section presents the evaluation method used in this work to study the overall performance and the sensitivity properties (as discussed earlier) of partitioning algorithms. In addition, interactive graphics tools used in this work to facilitate the evaluation and comparative studies of various partitioning algorithms are presented.

### *4.1. General strategy*

There is no established method for evaluating partitioning algorithms based on purely theoretical arguments. Therefore, the evaluation is generally done empirically on a computer. A simple and effective evaluation methodology is presented here. It uses a set of test problems to evaluate and compare various partitioning algorithms. These test problems can be divided into two general groups: one with both 2-D and 3-D regular grid problems, and the other with realistically irregular meshes found in a wide variety of finite element applications. Test problems in the first group are important for studying the characteristic behaviour and sensitivity properties of partitioning algorithms as the mesh is refined or the number of partitions varies in a systematic way. Test problems in the second group, which represent real problems to be solved in engineering practice, are also necessary for assessing the performance of partitioning algorithms. The test problems in this group are those that require parallel finite element analysis for their practical solutions. They are usually 'large' and may have complicated geometry.

In the present study, only a set of rectangular 2-D and 3-D grids with grid points equally spaced (see Figures 4(a) and 4(b), respectively) are used as the first group of test problems to demonstrate the proposed evaluation method. However, regular grids of other shapes, such as triangular, cylindrical, and spherical grids, may also be included in the evaluation method.

The second group of test problems should include a wide variety of practical examples. However, it is beyond the scope of this work to collect a set of standard test problems with all the desired features for comparative studies of partitioning algorithms. The model provided by the Harwell–Boeing sparse matrix collection[45] could be followed in future research to accomplish this task. In the present study, only two examples in the writers' library are used to demonstrate the proposed evaluation method. Although the mesh sizes of these examples cannot be considered large, their non-linear dynamic analyses do require parallel processing (see References 5 and 34). Furthermore, because the major interest in this work is to study the characteristic behaviour and

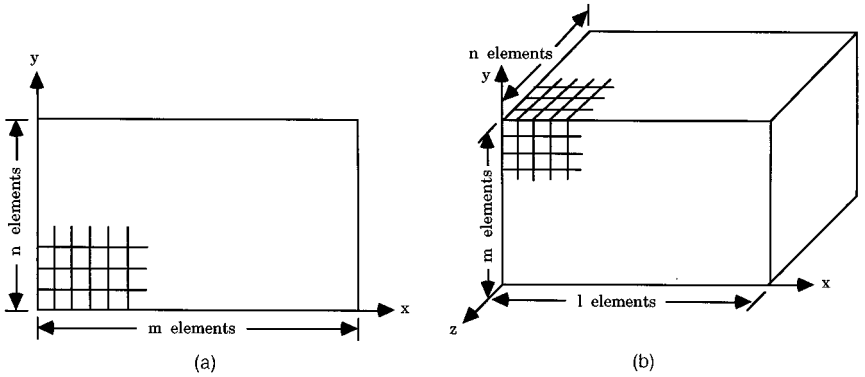


Figure 4. Regular grids problems used in the present study: (a) 2-D $m \times n$ grids; (b) 3-D $1 \times m \times n$ grids

sensitivity properties of partitioning algorithms using the proposed evaluation method, the characteristics of the meshes (such as multiply connected and/or branched domains) are more important than the sizes of the meshes, except for studying the efficiency of the partitioning algorithms. As will be discussed later in this section, the efficiency of the partitioning algorithms is not the major interest of this work.

### 4.2. Sensitivity properties explored

Some algorithms perform better in fine-grained partitioning than in coarse-grained partitioning, while some do just the opposite. To investigate how sensitive an algorithm is to the *granularity of partitions*, both small and large values of $N_p$ (relative to the problem size) should be used in partitioning the same test problems. Furthermore, in coarse-grained partitioning, a slight variation of $N_p$ on a given test problem may significantly affect the performance of some algorithms. To investigate the sensitivity of an algorithm to the *variation of* $N_p$, the present study uses the following values of $N_p$: 7, 8, 9, 32, and 128.

It is also necessary to vary the *mesh density* of a test problem to study its effect on the performance of an algorithm. The 2-D regular grid problems used in the present study include $10 \times 20$, $20 \times 20$, $30 \times 20$, $40 \times 20$, and $50 \times 20$ meshes, while the 3-D regular grid problems used include $5 \times 10 \times 10$, $10 \times 10 \times 10$, $15 \times 10 \times 10$, and $20 \times 10 \times 10$ meshes. However, based on the results obtained from the 2-D test problems, only a few 3-D test problems may be required for further investigation and/or verification.

In addition, as discussed earlier, performance of some algorithms may be affected by the *nodal or element numbering* of the finite element mesh. To study this effect, test problems used in the evaluation method should include the same meshes but with different nodal and/or element numbering. In the present study, $20 \times 10$ (vs. $10 \times 20$) and $10 \times 5 \times 10$ (vs. $5 \times 10 \times 10$) meshes are added in 2-D and 3-D test problems, respectively. In 2-D problems, for example, the nodes are numbered across the $y$ direction, which provides a regular numbering pattern. In this case, the $10 \times 20$ and $20 \times 10$ meshes are meshes of same topology and density but with different nodal and element numbering. Moreover, because the most natural numbering for bandwidth reduction is across the $x$ direction, the initial numbering (which is done across $y$ direction) ensures that algorithms depending on a good initial numbering would not have advantage over the others.

Furthermore, the effects of using finite elements of *different interpolation order* on the performance of an algorithm should be investigated. The present study uses Q4 and Q8 (eight-noded quadrilateral) elements for 2-D test problems, and B8 (eight-noded brick) and B20 (20-noded brick) elements for 3-D test problems. These elements have been selected because they are commonly used in finite element practice. However, different types of elements, such as triangular elements of 3 and 6 nodes for 2-D meshes, and tetrahedral elements of 4 and 10 nodes for 3-D meshes, could also be studied in the evaluation method.

### 4.3. Quantitative performance assessment

As discussed earlier in Section 2, different parallel solution methods may require different load balancing strategies to maximize their efficiencies. The focus of this work is on the element-based partitioning. Some commonly used evaluation criteria are considered in this work for assessing overall performance of partitioning algorithms in the following aspects: (a) balance of computational load among processors, (b) minimization of interprocess communication, (c) balance of interprocess communication, and (d) time required for partitioning algorithms. Specific evaluation parameters used in the numerical study are discussed below. It should be noted that

additional criteria may be needed if evaluation of partitioning algorithms for a specific parallel solution method is required.

- *Balance of computational load among processors*: All of the partitioning algorithms considered in this study are implemented assuming that the balance of computational load among processors is achieved by balancing the element distributions among subdomains before the partitioning process takes place (only meshes with elements of same type are considered here). As a result, the number of elements in any two subdomains differs at most by one for all cases. Since all algorithms perform equally well in balancing element distributions, no evaluation parameter is used for this aspect in the numerical study. However, it should be noted that although the above assumption may be appropriate for some parallel solution methods (see, for example, Reference 3), many parallel solution methods often require different element distribution schemes to achieve load balance. In this work, the focus is on the general framework of an evaluation method. The framework can then be tailored to meet the specific needs of a specific parallel solution algorithm. It can be seen later in Section 5 that the above assumption has little effect on the illustration of the evaluation methodology as well as the sensitivity studies of the partitioning algorithms.
- *Minimization of interprocess communication*: $\mathrm{Tot}(N_b)$, total number of boundary interface nodes in the domain, indicates total volume of message passing. $\mathrm{Ave}(N_a)$, average number of adjacent subdomains, shows the average number of messages needed for a process to communicate with its neighbours. $N_d$ is the number of subdomains that have disconnected regions. Intuitively, $N_d$ is a significant parameter because fragmented subdomains usually result in longer subdomain boundaries and more communication overhead. However, Hsieh and Abel[34] have shown that having non-fragmented subdomains (i.e. $N_d = 0$) is not sufficient in itself to obtain shorter subdomain boundaries.
- *Balance of interprocess communication*: The parameter used to evaluate the balancing of communication loads among processors is $\mathrm{Max}(N_b)/\mathrm{Min}(N_b)$, the ratio between the maximum and minimum number of subdomain boundary nodes among all subdomains.
- *Time required for partitioning algorithms*: The CPU time (s) required for partitioning on a single Sun SPARC 10/40 GX workstation is denoted as $T_{cpu}$ in this work. $T_{cpu}$ includes time spent in data preparation for the algorithm, execution of the algorithm, and setting results into the database. This is different from timing only execution of the core partitioning algorithm as usually adopted in the field of computer science but also by many other researchers. Different algorithms often have different input data requirements and produce partitioning results in different formats. Depending on the algorithm as well as the size and characteristics of the mesh, these pre- and post-processing phases may demand a dominant portion of the total CPU time required to obtain partitioning results for a parallel finite element program from conventional input data of a finite element mesh. Therefore, the CPU time presented in this work is a more practical measure of the complete partitioning task. However, because the partitioning algorithms studied are implemented by different researchers with different standards for efficiency optimization, it should be noted that the CPU time reported in this work might not reflect accurately the efficiency of the actual partitioning algorithms. In addition, the CPU time spent in partitioning is usually negligible compared to the execution time of parallel analysis.

### 4.4. Interactive graphics tools

An interactive graphics program called PSAINT[5,46] is used to facilitate the evaluation and comparative studies conducted in this work. All of the automatic domain partitioning algorithms compared in this paper have been implemented in PSAINT. A set of graphics tools is also provided in PSAINT for examination of partitioning results. For example, upon completion of

partitioning, the program displays subdomains in different colours. The users are then allowed to turn on and off display of any subdomain, to display subdomains individually, or to display them in a sequential order for better examination of partitioning results. In addition, message boxes containing statistics of partitioning results are automatically generated and displayed. Plate 1 shows the partitioning results of a 24-bladed disk model evaluated using PSAINT.

## 5. TEST PROBLEMS, RESULTS, AND DISCUSSIONS

Finite element problems of different types have been used to evaluate and compare the GR, PGR, ANP, RBD(Col), RBD(GPS), RST(DG), and RST(CG) partitioning algorithms. They include both 2-D and 3-D regular grid problems of different sizes and practical problems modelled by 3-D elements.

### *5.1. 2-D regular grid problems*

The type of 2-D regular grid problems considered in the present study is shown in Figure 4(a). The test problems include $20 \times 10$, $10 \times 20$, $20 \times 20$, $30 \times 20$, $40 \times 20$, and $50 \times 20$ meshes. The nodes are numbered across the $y$ direction. The values of $N_p$ studied are 7, 8, 9, 32, and 128. All meshes use either Q4 or Q8 elements. All of the numerical results have been summarized in Appendix B of Reference 47. Interested readers are welcome to obtain a copy of the results from the writers.

Based on the numerical results and graphical examination of these results in PSAINT, the following observations are made:

(1) The ANP, RBD(Col), and RBD(GPS) algorithms tend to produce stripwise partitions with longer boundaries but fewer neighbours, while the GR, PGR, RST(DG), and RST(CG) algorithms often generate domainwise partitions with smaller aspect ratios, shorter boundaries, but more neighbours. Furthermore, the GR and PGR algorithms often tend to generate a non-connected or stripwise subdomains in the last few partitions. The general characteristics of these partitioning algorithms are illustrated in Figure 5 using a $50 \times 20$ Q4 mesh with $N_p = 32$ (the partitioning result of the PGR algorithm is not shown because it is almost the same as that of the GR algorithm).

(2) The $10 \times 20$ and $20 \times 10$ meshes are meshes of same topology and density but with different nodal and element numbering. If the partitioning results of an algorithm are different on these two meshes, the algorithm is sensitive to the initial nodal and/or element numbering (from a numerical point of view). Therefore, from the results obtained for $10 \times 20$ and $20 \times 10$ meshes with Q4 or Q8 elements, it can be found that the GR and PGR algorithms are sensitive to the initial nodal and/or element numbering (see, for example, Figure 6) except for the case of $N_p = 8$. The results of the ANP algorithm is only affected by the initial nodal and/or element numbering for the case of $N_p = 32$ (see, for example, Figure 6(b)). The results of the RBD(Col) algorithm is also affected (see Figure 6) because the Collins algorithm fails to improve the initial nodal numbering of the $20 \times 10$ meshes for bandwidth reduction (in these cases, the initial numbering is kept) to be as good as the initial nodal numbering of the $10 \times 20$ mesh. Based on the results obtained for $10 \times 20$ and $20 \times 10$ meshes with Q4 or Q8 elements, the RBD(GPS) seems to be insensitive to the initial nodal numbering. However, problems similar to those discussed above for the RBD(Col) algorithm may also arise with the RBD(GPS) algorithm. In Figure 7, an example using $20 \times 30$ and $30 \times 20$ Q4 meshes with $N_p = 7$ is used to demonstrate this point. Theoretically, the

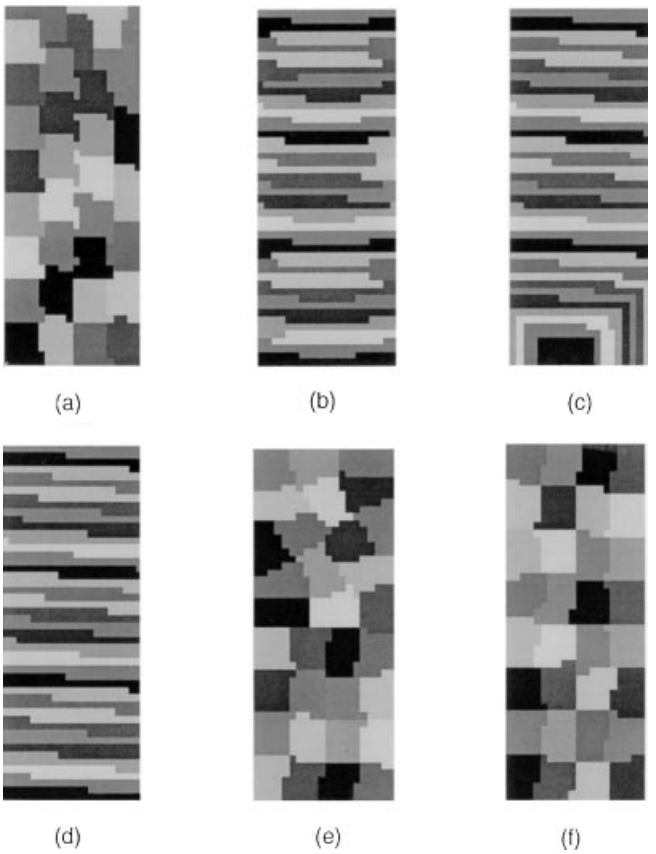


Figure 5. Partitioning of a $50 \times 20$ Q4 mesh ($Np = 32$) using: (a) GR; (b) ANP; (c) RBD(Col); (d) RBD(GPS); (e) RST(DG); (f) RST(CG) algorithms

RST algorithms should not be sensitive to the initial nodal or element numbering (or equivalently, to the initial vertex numbering of the associated graph). However, for the cases of $N_p = 32$ (see Figure 6(b)) and 128, their results are affected by element numbering. This may be due to the use of iterative solvers for approximating the eigensolutions, which are sensitive to the initial vertex numbering of the associated graph.

(3) Only the ANP algorithm is found to be sensitive to the element interpolation order as is seen from the results of the Q4 and Q8 meshes. In addition, the RST(DG) and RST(CG) algorithms become relatively less expensive (in terms of $T_{cpu}$) comparing with the non-spectral algorithms, such as the GR, PGR and RBD(GPS) algorithms, when higher-order elements are used. This is because the cost of the RST algorithms is a function of the number of elements in the mesh instead of the number of the nodes.

(4) The GR and PGR algorithms are sensitive to slight variations of $N_p$ (in this study, $N_p = 7$, 8 and 9). For example, consider the partitioning of the $40 \times 20$ Q4 mesh shown in Figure 8(a) using the GR algorithm. As shown in Figure 8(b), the algorithm produces optimal partitions when $N_p = 8$. However, for the cases of $N_p = 7$ and 9 (see Figures 8(c) and 8(d), respectively), one of the subdomains produced by the algorithm either is split into two parts

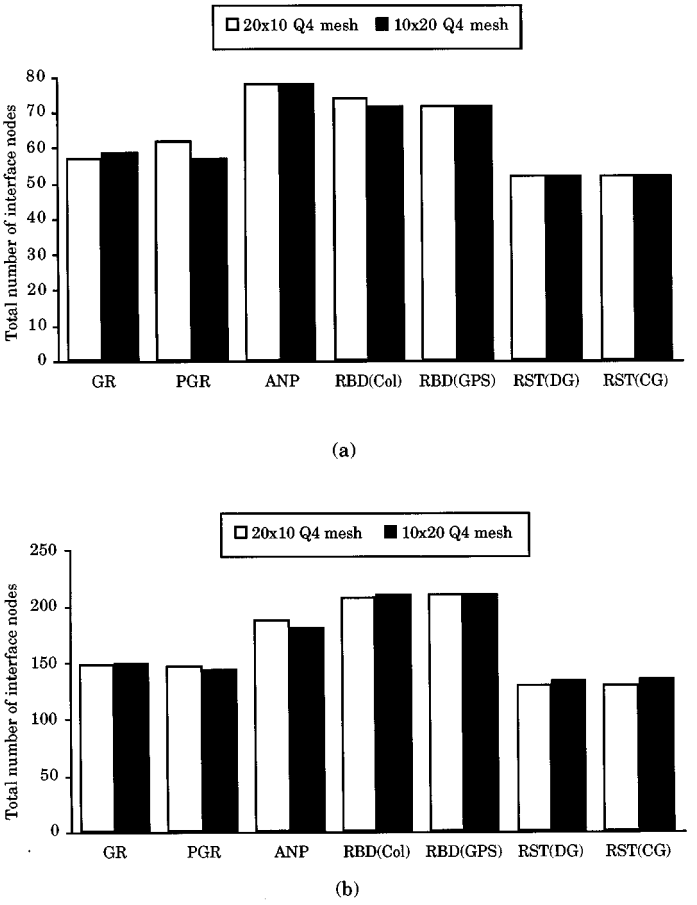


Figure 6. Comparison of partitioning results on $20 \times 10$ and $10 \times 20$ Q4 meshes for the cases: (a) $N_p = 7$; (b) $N_p = 32$

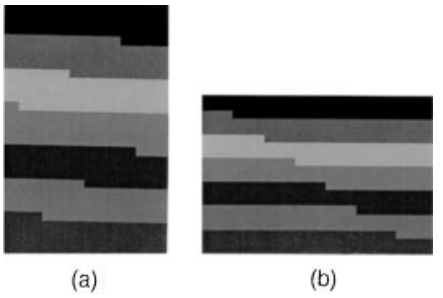


Figure 7. Partitioning results of the RBD(GPS) algorithm on: (a) $20 \times 30$; (b) $30 \times 20$ Q4 meshes ($N_p = 7$)

(for $N_p = 7$) or has a long boundary (for $N_p = 9$). Other algorithms seem to be less sensitive to this effect.

(5) The ANP and RBD algorithms are more likely to encounter domain-splitting problems for fine-grained partitioning (in this study, $N_p = 32$ and 128) than for coarse-grained

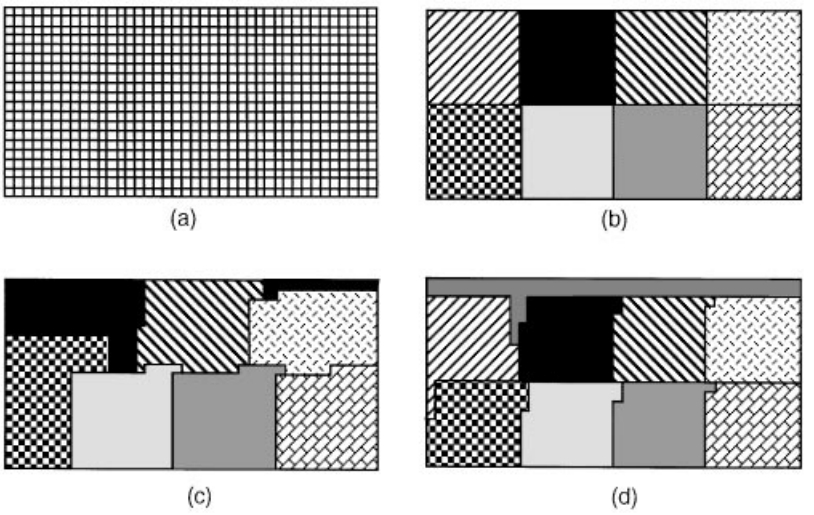


Figure 8. Partitioning of a $40 \times 20$ Q4 mesh using the GR algorithm: (a) $40 \times 20$ Q4 mesh; (b) $N_p = 8$; (c) $N_p = 7$; (d) $N_p = 9$

Table VI. Partitioning results ($N_p = 128$) on the 2-D $20 \times 20$ Q4 regular mesh (400 elements and 441 nodes)

| | | | | RBD | | RST | |
|---|---|---|---|---|---|---|---|
| Parameter | GR | PGR | ANP | (Col) | (GPS) | (DG) | (CG) |
| $T_{cpu}$ | 0·9 | 1·0 | 5·2 | 1·0 | 0·8 | 2·3 | 2·5 |
| Tot($N_b$) | 381 | 381 | 409 | 409 | 409 | 396 | 395 |
| Max($N_b$) | 8 | 8 | 12 | 10 | 10 | 10 | 10 |
| Min($N_b$) | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| Ave($N_a$) | 5·7 | 5·7 | 7·0 | 5·8 | 5·8 | 5·9 | 6·2 |
| $N_d$ | 0 | 0 | 16 | 11 | 11 | 0 | 0 |

partitioning (in this study, $N_p = 7$, 8 and 9). In addition, the GR and PGR algorithms sometimes outperform all other algorithms in the cases of fine-grained partitioning (see, for example, Table VI), but this is not the case for coarse-grained partitioning.

(6) All algorithms show some degree of sensitivity to the variation of mesh density. For example, the GR algorithm produces optimal solutions for the case of $N_p = 8$ on the $20 \times 10$, $10 \times 20$, and $40 \times 20$ meshes. However, it fails to do the same on the $20 \times 20$ meshes of Figure 9(a), as shown in Figure 9(b). This is not the case for both the RST(DG) and RST(CG) algorithms (see Figure 9(c)). In addition, for the $20 \times 20$ meshes having equal dimensions in $x$ and $y$ directions, the ANP algorithm does not utilize the geometrical information (i.e. no extra weight is added to the nodes) and produces the relatively poor results shown in Figure 9(d).

(7) The PGR algorithm does not always produce better results than the GR algorithm. However, it is a good alternative when the GR algorithm does not give satisfactory

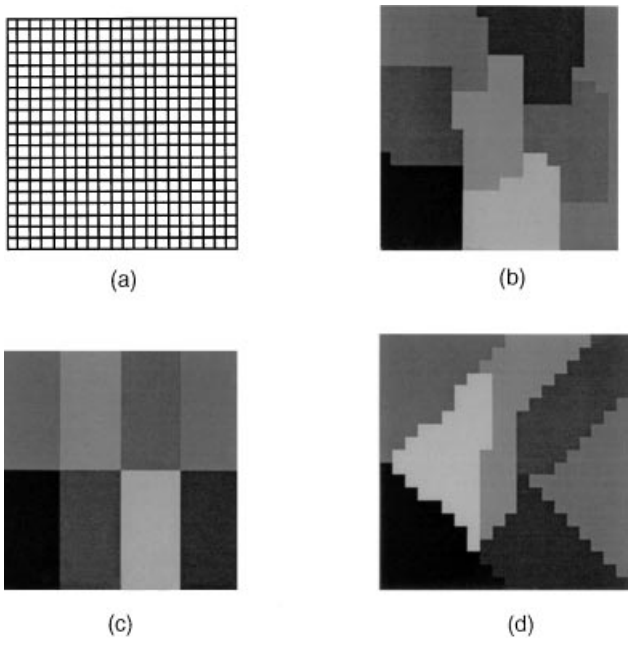


Figure 9. Partitioning of a $20 \times 20$ Q4 mesh ($N_p = 8$): (a) $20 \times 20$ Q4 mesh; (b) GR; (c) RST; (d) ANP

partitions. The comparison of results between the GR and PGR algorithms also suggest that the GR-type algorithms are sensitive to the selection of the very first node in the algorithm.

(8) The RST algorithms seem to perform the best among all those studied in terms of reducing both Tot($N_b$) and Ave($N_a$) in a balanced fashion. In addition, the RST algorithms are the only algorithms that do not have domain-splitting in this entire set of test problems.

### *5.2. 3-D regular grid problems*

The type of 3-D regular grid problems considered in the present study is shown in Figure 4(b). The test problems include $10 \times 5 \times 10$, $5 \times 10 \times 10$, $10 \times 10 \times 10$, $15 \times 10 \times 10$, and $20 \times 10 \times 10$ meshes. The nodes in the same $x$–$y$ plane are numbered across the $y$ direction and the nodes in the $x$–$y$ planes with smaller $z$ values are numbered before those with larger $z$ values. As in the 2-D cases, the values of $N_p$ studied are 7, 8, 9, 32 and 128. All meshes use B8 elements, while only the $5 \times 10 \times 10$, $10 \times 10 \times 10$, and $20 \times 10 \times 10$ meshes also use B20 elements. As in the 2-D cases, all of the numerical results are summarized in Appendix B of Reference 47. Interested readers are welcome to obtain a copy of the results from the writers.

The numerical results and graphical examination of these results in PSAINT further support the observations made in the 2-D regular grid problems. However, in this set of test problems, the RST(DG) algorithm also experiences domain-splitting situations in the cases of $20 \times 10 \times 10$ meshes with either B8 or B20 elements for $N_p = 128$. The RST(CG) thus becomes the only algorithm that does not have domain-splitting in both sets of regular grid problems.

### 5.3. *Practical problems*

The performance of the partitioning algorithms is further evaluated here using two practical examples consisting of solid elements. The first one, shown in Figure 10, is a turbine blade model with 944 B20 solid elements and 6427 nodes, while the second one, shown in Figure 11, is a 24-bladed turbine disk model with 720 B20 solid elements and 5688 nodes. In addition, other practical examples, such as those already shown in Figures 1–3, have been used in the examination of the GR, PGR, ANP, and RBD algorithms. Further numerical partitioning results of the

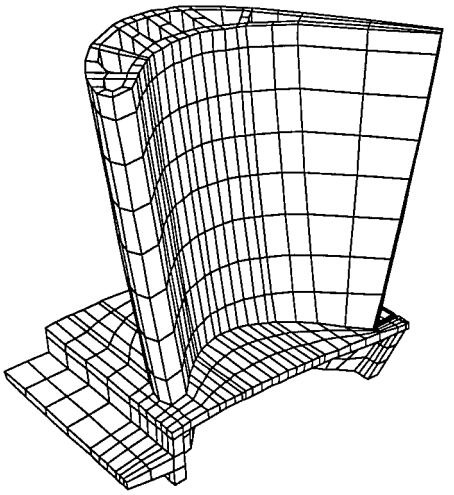

Figure 10. A finite element model of a turbine blade with 944 B20 elements and 6427 nodes[48]

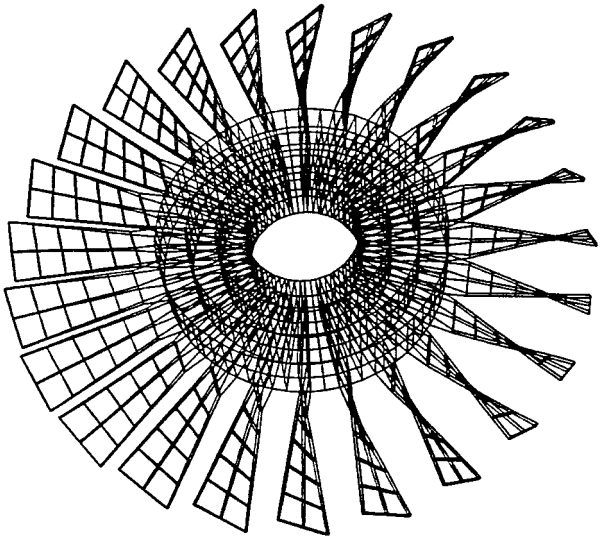

Figure 11. A finite element model of a 24-bladed disk with 720 B20 elements and 5688 nodes

RST(DG) and RST(CG) algorithms on several practical finite element examples can be found in the recent work by Hsieh *et al.*[6]

The partitioning results for the turbine blade problem are summarized in Table VII. It can be seen that the RST algorithms outperform all others and the RST(CG) algorithm is the only one that does not produce disconnected subdomains. Plates 2–4 show the partitioning results of the turbine blade problem with $N_p = 7$ for the GR, ANP, and RST(CG) algorithms, respectively. These plates use different views which, in the writers' judgment, illustrate each partitioning result in the best possible way.

The partitioning results for the 24-bladed turbine disk problem are summarized in Table VIII. Note that $N_p$ for this problem only includes 24 rather than 32 due to the 24-fold symmetry of this particular problem. It can be seen that the all algorithms including the RST(CG) algorithm have domain-splitting problems. However, the RST(CG) algorithm is the only algorithm that is able to furnish an optimal solution for the cases of $N_p = 8$ and 24 (see Plate 5 for the $N_p = 24$ case) for which it captures all the symmetries in the problem. Although the optimal partitioning solutions

Table VII. Partitioning results on the turbine blade model (944 elements and 6427 nodes)

| | | | | | RBD | | RST | |
|---|---|---|---|---|---|---|---|---|
| $N_p$ | Parameter | GR | PGR | ANP | (Col) | (GPS) | (DG) | (CG) |
| 7 | $T_{cpu}$ | 65·3 | 79·1 | 177·1 | 197·8 | 68·9 | 60·5 | 61·6 |
| | Tot($N_b$) | 1093 | 1113 | 1258 | 1204 | 1144 | 823 | 846 |
| | Max($N_b$) | 372 | 519 | 429 | 448 | 412 | 322 | 359 |
| | Min($N_b$) | 184 | 193 | 214 | 184 | 184 | 129 | 144 |
| | Ave($N_a$) | 4·3 | 4·0 | 4·6 | 1·7 | 1·7 | 3·4 | 3·4 |
| | $N_d$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | $T_{cpu}$ | 65·0 | 78·3 | 190·4 | 203·3 | 68·6 | 59·9 | 60·5 |
| | Tot($N_b$) | 1255 | 1164 | 1477 | 1487 | 1419 | 1004 | 955 |
| | Max($N_b$) | 491 | 468 | 539 | 524 | 484 | 363 | 308 |
| | Min($N_b$) | 174 | 197 | 197 | 174 | 174 | 128 | 138 |
| | Ave($N_a$) | 4·5 | 4·5 | 4·8 | 1·8 | 1·8 | 3·5 | 3·3 |
| | $N_d$ | 1 | 2 | 2 | 0 | 0 | 0 | 0 |
| 9 | $T_{cpu}$ | 64·1 | 78·5 | 191·3 | 196·7 | 71·4 | 60·4 | 60·9 |
| | Tot($N_b$) | 1213 | 1197 | 1466 | 1651 | 1510 | 1017 | 1107 |
| | Max($N_b$) | 420 | 452 | 397 | 549 | 477 | 353 | 340 |
| | Min($N_b$) | 159 | 178 | 185 | 194 | 159 | 113 | 143 |
| | Ave($N_a$) | 4·2 | 4·7 | 5·1 | 1·8 | 1·8 | 4·7 | 4·2 |
| | $N_d$ | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| 32 | $T_{cpu}$ | 68·7 | 81·3 | 208·3 | 196·8 | 70·2 | 62·1 | 64·0 |
| | Tot($N_b$) | 2370 | 2265 | 2596 | 4227 | 4121 | 2035 | 2134 |
| | Max($N_b$) | 223 | 205 | 247 | 330 | 326 | 198 | 199 |
| | Min($N_b$) | 79 | 79 | 68 | 83 | 81 | 79 | 78 |
| | *Ave*($N_a$) | 7·8 | 8·2 | 7·9 | 3·9 | 3·9 | 6·9 | 6·5 |
| | $N_d$ | 2 | 3 | 3 | 27 | 22 | 1 | 0 |
| 128 | $T_{cpu}$ | 71·6 | 83·5 | 316·6 | 202·9 | 75·0 | 66·0 | 67·6 |
| | Tot($N_b$) | 3827 | 3805 | 4343 | 4967 | 4783 | 3509 | 3801 |
| | Max($N_b$) | 104 | 109 | 120 | 125 | 110 | 98 | 92 |
| | Min($N_b$) | 33 | 43 | 33 | 49 | 33 | 33 | 33 |
| | Ave($N_a$) | 8·8 | 8·8 | 9·3 | 9·2 | 8·2 | 8·9 | 8·3 |
| | $N_d$ | 6 | 4 | 16 | 105 | 77 | 1 | 0 |

Table VIII. Partitioning results on the 24-bladed turbine disk (720 elements and 5688 nodes)

| | | | | | RBD | | RST | |
|---|---|---|---|---|---|---|---|---|
| $N_p$ | Parameter | GR | PGR | ANP | (Col) | (GPS) | (DG) | (CG) |
| 7 | $T_{cpu}$ | 43·9 | 54·5 | 118·4 | 125·7 | 47·2 | 41·7 | 40·1 |
| | Tot($N_b$) | 409 | 413 | 895 | 646 | 632 | 293 | 274 |
| | Max($N_b$) | 139 | 151 | 412 | 226 | 238 | 95 | 87 |
| | Min($N_b$) | 86 | 97 | 114 | 97 | 95 | 66 | 66 |
| | Ave($N_a$) | 2·6 | 2·9 | 2·3 | 1·7 | 1·7 | 2·3 | 2·0 |
| | $N_d$ | 4 | 5 | 1 | 6 | 7 | 3 | 2 |
| 8 | $T_{cpu}$ | 43·3 | 53·9 | 118·7 | 127·6 | 48·4 | 42·5 | 41·9 |
| | Tot($N_b$) | 425 | 412 | 876 | 770 | 695 | 327 | 296 |
| | Max($N_b$) | 131 | 136 | 327 | 220 | 228 | 103 | 74 |
| | Min($N_b$) | 79 | 79 | 135 | 110 | 71 | 58 | 74 |
| | Ave($N_a$) | 3·0 | 3·3 | 3·3 | 1·8 | 1·8 | 2·5 | 2·0 |
| | $N_d$ | 6 | 3 | 1 | 7 | 8 | 2 | 0 |
| 9 | $T_{cpu}$ | 44·1 | 54·9 | 122·9 | 126·2 | 47·6 | 42·2 | 41·7 |
| | Tot($N_b$) | 462 | 464 | 1025 | 802 | 784 | 395 | 309 |
| | Max($N_b$) | 123 | 162 | 363 | 207 | 220 | 113 | 74 |
| | Min($N_b$) | 84 | 79 | 111 | 97 | 79 | 66 | 66 |
| | Ave($N_a$) | 3·6 | 3·3 | 3·1 | 1·8 | 1·8 | 2·7 | 2·0 |
| | $N_d$ | 7 | 4 | 2 | 8 | 9 | 3 | 0 |
| 24 | $T_{cpu}$ | 45·0 | 56·3 | 145·7 | 127·2 | 48·5 | 43·8 | 42·6 |
| | Tot($N_b$) | 1169 | 1·72 | 1570 | 2500 | 2303 | 984 | 888 |
| | Max($N_b$) | 151 | 111 | 292 | 220 | 228 | 138 | 74 |
| | Min($N_b$) | 71 | 52 | 39 | 85 | 84 | 52 | 74 |
| | Ave($N_a$) | 3·8 | 3·5 | 4·1 | 1·9 | 1·9 | 3·1 | 2·0 |
| | $N_d$ | 17 | 12 | 4 | 23 | 24 | 7 | 0 |
| 128 | $T_{cpu}$ | 49·0 | 59·2 | 204·2 | 130·1 | 52·0 | 47·1 | 45·6 |
| | Tot($N_b$) | 2734 | 2647 | 2817 | 3819 | 3906 | 2463 | 2483 |
| | Max($N_b$) | 80 | 70 | 79 | 91 | 90 | 71 | 63 |
| | Min($N_b$) | 13 | 13 | 13 | 26 | 13 | 13 | 13 |
| | Ave($N_a$) | 5·1 | 4·6 | 6·1 | 5·0 | 5·7 | 4·5 | 4·4 |
| | $N_d$ | 21 | 13 | 12 | 87 | 101 | 8 | 4 |

in these two cases seem to be a trivial task for humans, this is obviously not the case for automatic partitioning algorithms. In addition, Plates 6–8 show the partitioning results of the 24-bladed turbine disk problem with $N_p = 9$ for the PGR, ANP, and RST(CG) algorithms, respectively.

Examples like the one displayed in Figure 11 with partitioning results in Plate 5 provide useful benchmarks for comparing partitioning algorithms. It is desirable that partitioning algorithms be able to cope with this type of mesh, which is branched and presents many symmetries.

## 6. CONCLUSIONS AND FUTURE DIRECTIONS

Based on the discussions and the numerical comparative studies presented above, the conclusions of this work are summarized as follows:

(1) The ANP, RBD(Col), and RBD(GPS) algorithms tend to produce stripwise partitions with longer boundaries but fewer neighbours, while the GR, PGR, RST(DG), and

RST(CG) algorithms often generate domainwise partitions with better aspect ratios, shorter boundaries, but more neighbours. Very recently, Farhat *et al.*[49] have discussed that most domain-decomposition-based parallel iterative solution schemes require partitions with low aspect ratios. Therefore, the ANP, RBD(Col) and RBD(GPS) algorithms are not suitable for those types of parallel solution schemes.

(2) The GR and PGR algorithms often tend to generate disconnected or stripwise subdomains in the last few partitions.

(3) From a numerical point of view, all algorithms studied in this work are sensitive to the initial nodal and/or element numbering. Therefore, preprocessing with a simple and efficient nodal-reordering algorithm is recommended for improved performance of the automatic partitioning algorithms.

(4) Only the ANP algorithm is found to be sensitive to the finite element interpolation order (e.g. Q4 vs. Q8 meshes).

(5) The GR and PGR algorithms are sensitive to slight variations of $N_p$ (in this study, $N_p = 7$, 8 and 9), while other algorithms seem to be less sensitive to this effect.

(6) The ANP and RBD algorithms are more likely to encounter domain-splitting problems for fine-grained partitioning (in this study, $N_p = 32$ and 128) than for coarse-grained partitioning (in this study, $N_p = 7$, 8 and 9). In addition, the GR and PGR algorithms sometimes outperform all other algorithms in the cases of fine-grained partitioning, but this is not the case for coarse-grained partitioning.

(7) In general, the RBD(GPS) algorithm is more effective than the RBD(Collins) algorithm because the GPS algorithm is more effective for bandwidth reduction than the Collins algorithm.

(8) The PGR algorithm is a good alternative when the GR algorithm does not give satisfactory partitions, and vice versa. Furthermore, both algorithms are sensitive to the selection of the very first node in the algorithm.

(9) All algorithms show some degree of sensitivity to the variation of mesh density.

(10) For problems where the longest and shortest Cartesian dimensions are equal, the ANP algorithm does not utilize the geometrical information, i.e. no extra weight is added to the nodes. For this special case, the ANP is not recommended, even for coarse-grained partitioning, because it often generates non-connected subdomains.

(11) For the GR, PGR, ANP and RBD algorithms, the domain-splitting situations occur mostly when the current partition reaches the boundary of the whole domain. This particular occurrence of domain splitting is typical for algorithms which use only local information in the mesh.

(12) Domain splitting is a common problem to all the domain-partitioning algorithms that have been investigated. Amongst the algorithms examined, the RST algorithm is the one with the least-frequent occurrence of splitting problems. The spectral approach to this problem is promising because of existing theoretical support.[22, 23, 50]

(13) The RST algorithms seem to perform the best among all those studied in terms of reducing both the total number of subdomain boundary nodes and the average number of adjacent subdomains in a balanced fashion. For fine-grained partitioning, the GR and PGR algorithms can sometimes be very effective and efficient in terms of minimizing both Tot($N_b$) and Ave($N_a$) although $N_d$ is often not zero.

(14) The evaluation method proposed in this work is useful and effective in evaluating different types of automatic domain partitioning algorithms. The test problems and results shown in this work also serve as benchmark examples for other researchers to evaluate both new and existing automatic partitioning algorithms. Furthermore, interactive graphics

programs such as PSAINT are essential for examination of partitioning results, especially for 3-D problems.

There are still several related issues that need to be addressed or further investigated and have not been done in this work. These issues are briefly discussed below:

(a) A standardized evaluation method with a set of standard but expandable benchmark examples will be useful for evaluation and comparative studies of partitioning algorithms. Work is needed in standardization of evaluation methods and especially in collecting benchmark test problems.
(b) The time required for the partitioning algorithms, as reported here, provides a measure of the total time spent in the complete mesh partitioning process, which includes some pre-processing (e.g. building appropriate graph data structure), actual partitioning, and some post-processing (e.g. setting results into the database). This timing is useful from a practical and qualitative points of view. However, in order to evaluate time complexity of the partitioning algorithms themselves, the pre- and post-processing stages should not be considered. Probably, the best metric for evaluating partition quality is the run-time of a parallel application. Nevertheless, this raises a number of issues associated with implementation details and machine/architecture selection. Development of appropriate metrics for partition quality is an important topic in the field of parallel computing. This is currently under investigation.
(c) Other important classes of domain partitioning algorithms that have not been evaluated in this work should be investigated. For example, the recent work by Hendrickson and Leland at Sandia National Laboratories is worth noting. They have developed a software package called Chaco,[51] which implements several algorithms for graph partitioning (e.g. spectral, inertial, Kernighan-Lin, and multilevel algorithms). In Reference 52, they have shown that the multilevel algorithm[53] can produce high-quality partitions similar to those obtained from the spectral algorithms, but at a much lower cost. The partitioning algorithms in Chaco, especially the multilevel one, should be further evaluated. More recently, Karypis and Kumar[54] have experimented with a class of multilevel algorithms. After investigating the effectiveness of many different choices for all three partitioning phases, they propose a new multilevel algorithm. In Reference 54, they have shown that their algorithm is faster than the multilevel algorithm in Chaco and produce partitions of similar high quality. Evaluation of this new algorithm should be included in the future study. In addition, the genetic algorithm by Topping and Khan,[55–57] and the Neural Networks algorithm by Fox[26] should be investigated.
(d) Greedy-type algorithms (e.g. the GR and PGR algorithms) are sensitive to the selection of the starting node (cf. Figures 1(b) and 1(c)). Therefore, it may be advantageous to use the

Table IX. Proposed modified PGR (MPGR) partitioning algorithm

| |
|---|
| (1) Use the adjacency structure of the non-zero weighted boundary nodes of the previously defined subdomain (for the first time, use the adjacency structure of the whole domain) to find a pseudoperipheral node. |
| (2) Assign unassigned elements that are connected to this node to the current subdomain. Recursively, assign unassigned elements that are adjacent to the elements in the current subdomain to the current subdomain until the number of elements equals to the total number of elements divided by the number of processors. |
| (3) Repeat (1) and (2) until all subdomains are defined. |

*pseudoperipheral* node concept during each stage of the iterative process of the PGR algorithm. Table IX presents the Modified PGR (MPGR) algorithm for this approach. The MPGR algorithm needs to be implemented and evaluated.

## APPENDIX

*Abbreviations of the partitioning algorithms*

Abbreviations of the partitioning algorithms used in this paper are summarized here. References for the algorithms are also indicated.

| | |
|---|---|
| 1DTF | 1D Topology Frontal algorithm[25] |
| ANP | Al-Nasra and Nguyen Partitioning algorithm[2] |
| ERB* | Eigenvector Recursive Bisection algorithm[27] |
| GR | GReedy algorithm[1] |
| MPGR | Modified Preconditioned GReedy algorithm† |
| ORB | Orthogonal Recursive Bisection algorithm[26, 27] |
| PGR | Preconditioned GReedy algorithm† |
| PI | Principal Inertia algorithm[11, 16, 25] |
| RBD(Col) | Reduced Bandwidth Decomposition (Collins) algorithm[3] |
| RBD(GPS) | Reduced Bandwidth Decomposition (GPS) algorithm† |
| RCB | Recursive Coordinate Bisection algorithm[4, 24] |
| RCM‡ | Reverse Cuthill–McKee-based algorithm[25] |
| RGB | Recursive Graph Bisection algorithm[4,24] |
| RPI | Recursive Principal Inertia algorithm[11, 16, 25] |
| RRCM | Recursive Reverse Cuthill–McKee algorithm[25] |
| RSB* | Recursive Spectral Bisection algorithm[4, 24] |
| RSO | Recursive Spectral Octasection algorithm[14, 15] |
| RSQ | Recursive Spectral Quadrisection algorithm[14, 15] |
| RSS | Recursive Spectral Sequential-cut algorithm[5, 6] |
| RST | Recursive Spectral Two-way algorithm[5, 6] |
| SA | Simulated Annealing algorithm[9, 10, 26, 27] |

ACKNOWLEDGMENTS

The writers wish to thank Dr. Sanjeev Srivastav of Stress Technology Inc. and formerly of Cornell University for help in the development of the program PSAINT, Dr. Horst D. Simon of NASA Ames Research Center for kindly providing the source code of the fast multilevel implementation of the RSB algorithm, Prof. Charbel Farhat of University of Colorado at Boulder for providing corrections to the published version of the GR algorithm, and Dr. Charles Lawrence of NASA Lewis Research Center for providing a sample mesh model for creation of the turbine disk example of Figure 11. The support of NASA Lewis Research Center under Grant No. NAG 3-1063 has been essential to this work and is appreciated. In addition, Shang-Hsien Hsieh would

* The ERB and RSB algorithms are essentially the same
† Present work
‡ Note that the RCM reordering algorithm also has the same abbreviation

like to acknowledge the partial support of U.S. Department of Energy under Award No. DE-FG02-93ER25169. Glaucio H. Paulino would like to acknowledge the partial financial support provided by the Brazilian agency CNPq (National Council for Research and Development). Finally, the writers acknowledge an anonymous reviewer for helpful comments.

## REFERENCES

1. C. Farhat, 'A simple and efficient automatic FEM domain decomposer', *Comput. Struct.*, **28,** 579–602 (1988).
2. M. Al-Nasra and D. T. Nguyen, 'An algorithm for domain decomposition in finite element analysis', *Comput. Struct.*, **39,** 277–289 (1991).
3. J. G. Malone, 'Automatic mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computers', *Comput. Methods Appl. Mech. Eng.*, **70,** 27–58 (1988).
4. H. D. Simon, 'Partitioning of unstructured problems for parallel processing', *Comput. Systems Eng.*, **2,** 135–148 (1991)
5. S. H. Hsieh, 'Parallel processing for nonlinear dynamics simulations of structures including rotating bladed-disk assemblies', *Ph.D. Dissertation*, Cornell University, Ithaca, New York, 1993.
6. S. H. Hsieh, G. H. Paulino and J. F. Abel, 'Recursive spectral algorithms for automatic domain partitioning in parallel finite element analysis', *Comput. Methods Appl. Mech. Eng.*, **121,** 137–162 (1995).
7. P. Le Tallec, 'Domain decomposition methods in computational mechanics', *Comput. Mech. Adv.*, **1,** 121–140 (1994).
8. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to The Theory and NP-Completeness*, W. H. Freeman and Company, New York, 1979.
9. J. Flower, S. Otto and M. Salama, 'Optimal mapping of irregular finite element domains to parallel processors', in A. K. Noor (ed.), *Parallel Computations and their Impact on Mechanics*, AMD-Vol. 86, ASME, New York, 1987, pp. 239–252.
10. B. Nour-Omid, A. Raefsky and G. Lyzenga, 'Solving finite element equations on concurrent computers', in A. K. Noor (ed.), *Parallel Computations and their Impact on Mechanics*, AMD-Vol. 86, ASME, New York, 1987, pp. 209–228.
11. C. Farhat and M. Lesoinne, 'Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics', *Int. j. numer. methods eng.*, **36,** 745–764 (1993).
12. F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
13. J. G. Malone, 'Parallel nonlinear dynamic finite element analysis of three-dimensional shell structures', *Comput. Struct.*, **35,** 523–539 (1990).
14. B. Hendrickson and R. Leland, 'An improved spectral partitioning algorithm for mapping parallel computations', *Sandia Report SAND92-1460*, Category UC-405, Sandia National Laboratories, Albuquerque, NM 87185, 1992.
15. B. Hendrickson and R. Leland, 'Multidimensional spectral load balancing', *Sandia Report SAND93-0074*, Category UC-405, Sandia National Laboratories, Albuquerque, NM 87185, 1993.
16. C. Farhat and M. Lesoinne, 'Mesh partitioning algorithms for the parallel solution of partial differential equations', *Appl. Numer. Math.*, **12,** 443–457 (1993).
17. J. Rodriguez and J. Sun, 'A domain decomposition study for a parallel finite element code', in: G. A. Gabriele (ed.), *Computers in Engineering*, Vol. 2, ASME, New York, 1992, pp. 83–90.
18. J. Padovan and A. Kwang, 'Hierarchically parallelized constrained nonlinear solvers with automated substructuring', *Comput. Struct.*, **41,** 7–33 (1991).
19. G. L. Miller, S.-H. Teng, W. Thurston and S. A. Vavasis, 'Automatic mesh partitioning', *Technical Report CTC92TR112*, Cornell Theory Center, Cornell University, Ithaca, New York, 1992.
20. E. R. Barnes, 'An algorithm for partitioning the nodes of a graph', *SIAM J. Algebraic Discrete Methods*, **3,** 541–550 (1982).
21. A. Pothen, H. D. Simon and K.-P. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl.*, **11,** 430–452 (1990).
22. W. N. Anderson Jr. and T. D. Morley, 'Eigenvalues of the Laplacian of a Graph', *Linear Multilinear Algebra*, **18,** 141–145 (1985) (originally published as *University of Maryland Technical Report TR-71-45*, October 1971).
23. M. Fiedler, 'Algebraic connectivity of graphs', *Czechoslovak Math. J.*, **23,** 298-305 (1973).
24. V. Venkatakrishnan, H. D. Simon and T. J. Barth, 'A MIMD implementation of a parallel Euler solver for unstructured grids', *J. Supercomput.*, **6,** 117–137 (1992).
25. C. Farhat and H. D. Simon, 'TOP/DOMDEC: a software tool for mesh partitioning and parallel processing', *Report CU-CSSC-93-11*, Center for Space Structures and Controls, College of Engineering, University of Colorado, Boulder, CO 80309, 1993.
26. G. C. Fox, 'A review of automatic load balancing and decomposition methods for the hypercube', in M. Schultz (ed.), *Numerical Algorithms for Modern Parallel Computer Architectures—The IMA Volumes in Mathematics and its Applications*, Vol. 13, Springer, New York, 1988, pp. 63–76.
27. R. D. Williams, 'Performance of dynamic load balancing algorithms for unstructured mesh calculations', *Concurrency: Practice Experience*, **3,** 457-481 (1991).
28. S. H. Hsieh and G. H. Paulino, 'An evaluation methodology for automatic mesh partitioning algorithms', in B. H. V. Topping (ed.), *Advances in Computational Structures Technology*, Civil-Comp Press, Edinburgh, 1996, pp. 247–252.

29. C. Farhat, personal communication, 1992.
30. R. J. Collins, 'Bandwidth reduction by automatic renumbering', *Int. j. numer. method eng.*, **6,** 345–356 (1973).
31. S. T. Barnard and H. D. Simon, 'A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems', in: R. F. Sincovec *et al.* (eds.), *Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1993, pp. 711–718.
32. J. F. Hajjar and J. F. Abel, 'Parallel processing for transient nonlinear structural dynamics of three-dimensional framed structures using domain decomposition', *Comput. Struct.*, **30,** 1237–1254 (1988).
33. J. F. Hajjar and J. F. Abel, 'Parallel processing of central difference transient analysis for three-dimensional nonlinear framed structures', *Commun. Appl. Numer. Methods*, **5,** 39–46 (1989).
34. S. H. Hsieh, and J. F. Abel, 'Use of networked workstations for parallel nonlinear structural dynamic simulations of rotating bladed-disk assemblies', *Comput. Systems Eng.*, **4,** 521–530 (1993).
35. G. H. Paulino, I. F. M. Menezes, M. Gattass and S. Mukherjee, 'A new algorithm for finding a pseudoperipheral vertex or the endpoints of a pseudodiameter in a graph', *Commun. Numer. Methods Eng.*, **10,** 913–926 (1994).
36. J. A. George and J. W.-H. Liu, 'An implementation of a pseudoperipheral node finder', *ACM Trans. Math. Software*, **5,** 284–295 (1979).
37. N. E. Gibbs, W. G. Poole Jr. and P. K. Stockmeyer, 'An algorithm for reducing the bandwidth and profile of a sparse matrix', *SIAM J. Numer. Anal.*, **13,** 236–250 (1976).
38. E. Cuthill and J. McKee, 'Reducing the bandwidth of sparse symmetric matrices', *Proc. 24th National Conf.*, ACM Publication P-69, 1969, pp. 157–172.
39. W. M. Chan and A. George, 'A linear time implementation of the reverse Cuthill–Mckee algorithm', *BIT*, **20,** 8–14 (1980).
40. J. Puttonen, 'Simple and effective bandwidth reduction algorithm', *Int. j. numer. methods eng.*, **19,** 1139–1152 (1983).
41. W. H. Liu and A. H. Sherman, 'Comparative analysis of the Cuthill–Mckee and the reverse Cuthill–Mckee ordering algorithms for sparse matrices', *SIAM J. Numer. Anal.*, **13,** 198–213 (1976).
42. B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, N.J., 1980.
43. C. C. Paige and M. A. Saunders, 'Solution of sparse indefinite systems of linear equations', *SIAM J. Numer. Anal.*, **12,** 617–629 (1974).
44. B. Kernighan and S. Lin, 'An efficient heuristic procedure for partitioning graphs', *Bell System Tech. J.*, **29,** 291–307 (1970).
45. I. S. Duff, R. G. Grimes and J. G. Lewis, 'Sparse matrix test problems', *ACM Trans. Math. Software*, **15,** 1–14 (1989).
46. S. H. Hsieh, 'A mesh partitioning tool and its applications to parallel processing', *Proc. 1994 Int. Conf. on Parallel and Distributed Systems*, Hsinchu, Taiwan, R.O.C., 19–21 December, 1994, pp. 168–173.
47. S. H. Hsieh, G. H. Paulino and J. F. Abel, 'Evaluation of automatic domain partitioning algorithms for parallel finite element analysis', *Structural Engineering Report 94-2*, School of Civil and Environmental Engineering, Cornell University, Ithaca, New York, 1994.
48. P. A. Wawrzynek, 'Discrete modeling of crack propagation: theoretical aspects and implementation issues in two and three dimensions', *Ph.D. Dissertation*, Cornell University, Ithaca, New York, 1991.
49. C. Farhat, N. Maman, and G. W. Brown, 'Mesh partitioning for implicit computations via domain decomposition: impact and optimization of the subdomain aspect ratio', *Report CU-CAS-94-02*, Center for Aerospace Structures, College of Engineering, University of Colorado, Boulder, CO 80309, 1994.
50. M. Fiedler, 'A property of eigenvectors of non-negative symmetric matrices and its application to graph theory', *Czechoslovak Math. J.*, **25,** 619–633 (1975).
51. B. Hendrickson and R. Leland, 'The Chaco user's guide, version 1.0', *Technical Report SAND93-2339*, Sandia National Laboratories, Albuquerque, NM 87185, 1993.
52. R. Leland and B. Hendrickson, 'An empirical study of static load balancing algorithms', *Proc. Scalable High-Performance Computing Conference*, Knoxville, Tennessee, 23–25 May, 1994, pp. 682–685.
53. B. Hendrickson and R. Leland, 'A multilevel algorithm for partitioning graphs', *Technical Report SAND93-1301*, Sandia National Laboratories, Albuquerque, NM 87185,1993.
54. G. Karypis and V. Kumar, 'A fast and high quality multilevel scheme for partitioning irregular graphs', *Technical Report 95-035*, Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, 1995.
55. B. H. V. Topping and A. I. Khan, 'An optimization-based approach to the domain decomposition problem in parallel finite element analysis', in M. P. Bendsøe and C. A. M. Soares (eds.), *Topology Design of Structures*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993, pp. 535–546.
56. A. I. Khan and B. H. V. Topping, 'Subdomain generation for parallel finite element analysis', *Comput. Systems Eng.*, **4,** 473–488 (1993).
57. B. H. V. Topping and A. I. Khan, 'Sub-domain generation method for non-convex domains', in B. H. V. Topping and A. I. Khan (eds.), *Information Technology for Civil and Structural Engineers*, Civil-Comp Press, Edinburgh, U.K., 1993, pp. 219–234.