Structural Engineering and Mechanics, Vol. 14, No. 6 (2002) 625-647
DOI: http://dx.doi.org/10.12989/sem.2002.14.6.625

Automatic decomposition of unstructured meshes employing genetic algorithms for parallel FEM computations

A. Rama Mohan Rao† and T.V.S.R. Appa Rao‡

Structural Engineering Research Centre, CSIR Campus, Taramani, Chennai 600 113, India

B. Dattaguru‡†

Department of Aerospace Engineering, Indian Institute of Science, Bangalore 560 012, India

(Received December 11, 2001, Accepted October 9, 2002)

Abstract. Parallel execution of computational mechanics codes requires efficient mesh-partitioning techniques. These mesh-partitioning techniques divide the mesh into specified number of submeshes of approximately the same size and at the same time, minimise the interface nodes of the submeshes. This paper describes a new mesh partitioning technique, employing Genetic Algorithms. The proposed algorithm operates on the deduced graph (dual or nodal graph) of the given finite element mesh rather than directly on the mesh itself. The algorithm works by first constructing a coarse graph approximation using an automatic graph coarsening method. The coarse graph is partitioned and the results are interpolated onto the original graph to initialise an optimisation of the graph partition problem. In practice, hierarchy of (usually more than two) graphs are used to obtain the final graph partition. The proposed partitioning algorithm is applied to graphs derived from unstructured finite element meshes describing practical engineering problems and also several example graphs related to finite element meshes given in the literature. The test results indicate that the proposed GA based graph partitioning algorithm generates high quality partitions and are superior to spectral and multilevel graph partitioning algorithms.

Key words: load balancing; mesh partitioning; genetic algorithms; multilevel approaches; unstructured meshes; parallel processing; dual graph.

1. Introduction

Complex engineering problems, which are modelled through partial differential equations, require large size meshes and the solution of these problems demand enormous computer resources in terms of both memory and CPU. Hence, they are often too large to solve on sequential computers, because of either memory limitations or computational demands, or both. Parallel processing is one of the cost-effective ways to overcome this situation. In order to solve these problems on distributed

† Scientist
‡ Former Director
‡† Professor and Chairman

memory parallel systems, the data need to be partitioned and sent to the processing units, so that the computational work load among the processors is approximately equal and at the same time the data exchange among the processors is minimum.

The problem of finite element mesh decomposition is equivalent to partitioning the deduced graph of the finite element mesh into subgraphs of roughly equal size such that the partitions cut the least number of edges of the graph. The n-way graph partitioning problem can be defined as follows: Let $G = G(N, E)$ be an undirected graph where N is the set of vertices with $\|N\|$ vertices and E is the set of edges with $\|E\|$ edges, partition N into n subsets, $N_1, N_2, N_3, \ldots \ldots N_n$ such that $N_i \cap N_j = 0$ for $i \neq j$, $\|N_i\| = \|N\|/n$ and $U_i\, N_i = N$, and the number of edges of E whose incident vertices belong to different subsets is minimised. The problem of graph partitioning is well known in the graph theory literature and is not solvable in polynomial time. It is in fact classified as a *NP*-hard problem (Garey and Johnson 1979). Fortunately, it is not necessary to find an optimal solution to the problem, as a good quality sub-optimal partition is usually adequate.

The n-way partitioning problem is generally solved by recursive bisection or by K-way partitioning. A great deal of effort has been invested in studying the graph partitioning problem and as a result, a large number of successful partitioning methods have been developed. These include methods based on heuristic searches most notably the Farhat's greedy (Farhat 1988), and Kernighan-Lin heuristics (Kernighan and Lin 1970), coordinate based bisections (Jones and Plassmann 1994), inertial methods (De Keyser and Roose 1992), methods based on geometric partitioning (Miller *et al.* 1998) such as graph growing algorithms, techniques employing neural networks (Rama Mohan Rao *et al.* 1998, Pain *et al.* 1999), simulated annealing (Williams 1991, Bouhmala and Pahud 1998), graph bisection algorithms like spectral bisection algorithms (Simon 1991, Barnard and Simon 1994) and finally multilevel-graph bisection algorithms (Karypis and Vipin Kumar 1999, Hendrickson and Leland 1995, Walshaw and Cross 1999) which usually combine a graph contraction algorithm with a coarse graph partitioning algorithm and local refinement algorithms. Spectral bisection algorithm has established a reputation for producing high-quality partitions. Since spectral bisection algorithm requires calculation of the (Fiedler vector) eigenvector associated with the second largest eigenvalue of a Laplacian matrix, it is computationally expensive. In view of this, these spectral algorithms are usually combined with multilevel algorithms (Barnard and Simon 1994, Hendrickson and Leland 1995) to produce fast and high quality graph partitions. A detailed review of these techniques is given elsewhere (Rama Mohan Rao 2001).

The multilevel algorithms (Karypis and Vipin Kumar 1999, Hendrickson and Leland 1995, Walshaw and Cross 1999) which permit to use any of the partitioning methods discussed earlier, emerged as faster and qualitatively superior graph partitioning algorithms. In these multilevel algorithms, the coarse graph representations of finer graphs are typically obtained by vertex collapsing (Karypis and Vipin Kumar 1999). Partitions on the coarsest graph are quickly obtained even when relatively expensive partitioning methods are used. These partitions are then mapped onto a finer mesh after which typically local searches (Kernighan-Lin heuristics or its variances) are performed to improve the partition quality, before proceeding to map the resulting partition onto the next finest graph or so on until target graph is partitioned. These algorithms are found to be extremely fast. In this paper, development and implementation of a new partitioning algorithm based on these multilevel concepts is presented. The proposed algorithm has been devised employing traditional binary encoded genetic algorithms.

2. Genetic algorithms

Genetic Algorithms (GA), first conceived by Holland (Holland 1975) in the 1970s, have received substantial attention during the last decade and have been applied in engineering domains (design, simulation, synthesis etc.) as well as many other domains (Altman *et al.* 1993, Mitchel *et al*. 1976, and Punch *et al*. 1993). Based on a set of probabilistic rules, GAs utilise the processes of natural selection by mimicking the concept of survival of the fittest. GAs are excellent all-purpose optimisation algorithms because they can accommodate both discrete and continuous valued design variables and search through nonlinear or noisy search spaces by using payoff (objective or cost function) information only. Genetic algorithms have number of unique features. First, GA does not search for a single solution, but infact maintains a set of perhaps thousands of solutions, which are termed as population. Second, GA attempts to increase the fitness of this population at each generation. Each solution is evaluated as to its fitness based on some domain specific function (usually the cost function). The solution is then kept or discarded based on that evaluation. If discarded, that member of the population is replaced by a new solution that is created by recombination of parts of existing good solutions. This process is repeated thousands, perhaps millions of times, combining different aspects of good solutions, and searching for a combination of solution features that is optimal under the evaluation function.

In genetic algorithms, a solution (i.e., a point in the search space) is called a chromosome, string, or genome. A typical GA approach requires population of chromosomes, each representing a combination of features from the set of features, and requires a cost function H. The algorithm manipulates a finite set (population) of chromosomes, based loosely on the mechanism of natural evolution. In each generation, chromosomes are subjected to certain operators, such as crossover, mutation, analogous to processes that occur in natural reproduction. The crossover of two chromosomes produces a pair of offspring chromosomes that are syntheses or combinations of the traits of their parents. A mutation on a chromosome produces nearly identical chromosome with only local alterations of some regions of the chromosome.

The optimisation process is performed in cycles called generations. During each generation, a set of new chromosomes is created using the GA operators like crossover and mutation. Since the population size is finite, only the best chromosomes are allowed to survive to the next cycle of reproduction. There is considerable variation among various implementations of the genetic algorithm approach in the strictness with which the principle of “survival of the fittest” is applied. In some systems, the fitness affects only the probability of survival, whereas in others, only the N most fit individuals are allowed to survive in each generation. The crossover rate is usually assumed as quite high values (in the order of 80 to 90%), while mutation rate is small (typically upto 15%) for efficient search (Goldberg and Segrest 1997). The cycle repeats until the population converges i.e., all the solutions are reasonably same and further exploration seems pointless or until the answer is good enough. Detailed discussion on these Genetic Algorithms can be found in reference (Goldberg 1989). Hence, in order to apply Genetic Algorithms one need to arrive at the objective function and one must choose the most appropriate representation of genome.

Literature on mesh-partitioning techniques employing genetic algorithms is scanty. Khan and Topping (1998) have used genetic algorithms for partitioning finite element meshes. However, rather than explicitly representing the partition, their approach used a population of cutting planes which bisected the finite element domain. A well-balanced partition is not sought by the technique, since it was designed for short run-times and thus used as an estimation of number of elements to appear in final refined sub-meshes. Mansoor and Fox (1991, 1994), partitioned graphs with a genetic algorithm using a direct encoding, where the sub-mesh membership of each vertex was explicitly represented by the value of a gene. Since these values were unconstrained, partitions of arbitrary imbalance were possible. These genes were concatenated and subjected to two-point crossover. The imbalance constraint was progressively enforced during evolution through the use of a penalty term in the fitness function. Wendl (1996) has devised a seed based decomposition procedure employing parallel genetic algorithms and concluded that the results of the GA based algorithm are comparable to simulated annealing but lacks consistency in providing continuously optimal solutions. Genetic algorithm using the direct coding has also been applied by Gil *et al.* (1998) to graph partitioning in the context of circuit partitioning. However, it was reported by the authors that GA based approach was outperformed by an approach devised synthesizing simulated annealing with tabu search. Soper, Walshaw and Cross (2000) have devised a graph partitioning technique by combining a multilevel algorithm with an evolutionary search procedure. The results reported are quite superior to all the state-of-the-art mesh partitioning algorithms.

In this paper, a new graph partitioning algorithm is proposed and it also combines multilevel scheme with GA. However, our algorithm is distinctly different from the work of Soper, Walshaw and Cross (2000). In the proposed work, traditional GA has been employed to develop graph partitioning algorithm. The ideas related to graph coarsening and projection schemes popularly being used in multilevel algorithms are borrowed and utilised in our work to devise an efficient problem representation as outlined in the subsequent sections. In contrast, Soper *et al.* (2000) have proposed the evolutionary algorithm as a layer over their multilevel graph partitioning algorithm called JOSTLE (Walshaw and Cross 1999). Hence, the evolutionary algorithm will not perform graph partitioning. Instead, it helps in improving the quality of mesh partitions generated by JOSTLE. It can also be noticed that the evolutionary search algorithm of Soper et al. is not a traditional GA since no problem representation (or genotype) is maintained. The crossover and mutation operators are simulated to suit the JOSTLE multilevel graph partitioning algorithm. In the present work, traditional GA with the conventional selection, crossover and mutation operators has been employed to devise a graph partitioner with efficient problem representation (Genome construction). In view of these differences, obviously, the cost function employed in these two algorithms also differs. Finally, the proposed GA based graph partitioning algorithm is formulated as recursive bisection algorithm while the Soper's evolutionary algorithm is a *K*-way partitioning algorithm.

3. Problem formulation

In this paper, the problem of mesh partitioning has been formulated as recursive bisection which gives rise to 2^n submeshes. The genetic algorithms operate on the dual graph of the finite element mesh. Due to their recursive character, the problem can be formulated for a single bisection. By repetitively applying the same algorithm, 2^n partitions can be obtained. The proposed formulation is based on mapping of the dual graph vertices to a bit string or genome, which is suitable for selected optimizer (GAs). Each bit corresponds to a single graph vertex and its bit value determines whether this vertex belongs to the one or other partition. In genetic algorithms, its fitness score expresses the performance of each bit-string representing a candidate solution. A cost function which is built with the requirements of the mesh partitioning problem i.e., load balance and also minimum interface size, returns a single value (fitness score) for each bit string.

3.1 Cost function

The cost function for graph partitioning problem can be written as:

$$H = H_{calc.} + \mu H_{comm.} \tag{1}$$

Where H_{calc} represents load imbalance and is minimised when all the processors have the same load and H_{comm} measures the communication cost and μ represents a factor which can be used to indicate the relative importance of the two terms in that particular context.

For graph bisection, the first term can be written as:

$$H_{calc} = \frac{\left| |V_1| - |V_2| \right|}{\sqrt{|V_1| \cdot |V_2|}} \tag{2}$$

In addition, the second term, which represents the communication cost, can be written as:

$$H_{comm} = \frac{N_{ce}}{\min(|V_1|, |V_2|)} \tag{3}$$

Where N_{ce} is the total number of cut edges in the graph. V_1 and V_2 represent the number of vertices in each partition. The term H_{calc} become zero if the partitions are perfectly balanced. Among all possible solutions with $|V_1| = |V_2|$, the optimum solution is the one with minimum N_{ce}. In contrast to this, minimisation of the second term, which measures the interface length is not stand alone and is to be considered in conjunction with the first term as $N_{ce} = 0$ will lead to a meaningless solution.

The value of μ need to be adjusted according to the size of the mesh. A Low value of μ lead to perfectly balanced partitions with an optimum or near optimum number of edge cuts. On the other hand, a higher value of μ may reduce the edge cuts at the cost of some imbalance in partitions. In order to obtain optimal value for μ, several test graphs of varying size have been solved. The value of μ is adjusted for all these test graphs in such a way that the partitions are perfectly balanced (i.e., near zero imbalance) with optimal cut edges. The following expression has been arrived at based on the outcome of these studies

$$\mu \approx 10\,(5.5 - \ln |V|\,) \tag{4}$$

Where $|V|$ is the number of vertices in the graph.

3.2 GA operators

For efficient implementation of Genetic Algorithms, a judicious selection of the three basic operators i.e., selection, crossover and mutation are necessary. They need to be tuned if necessary to the needs of the specific problem. The parent selection operators use the fitness score to decide which of the individuals will be mated in order to produce offspring. These individuals are copied to the mating pool where the crossover and the mutation operators are applied.

Several crossover and mutation operators are reported in the literature and some of them are customised specifically to the application. A simple one point crossover takes two individuals and cuts their chromosome strings at some randomly chosen position to produce two 'head' segments and two 'tail' segments. The tail segments are then swapped over to produce two new full-length

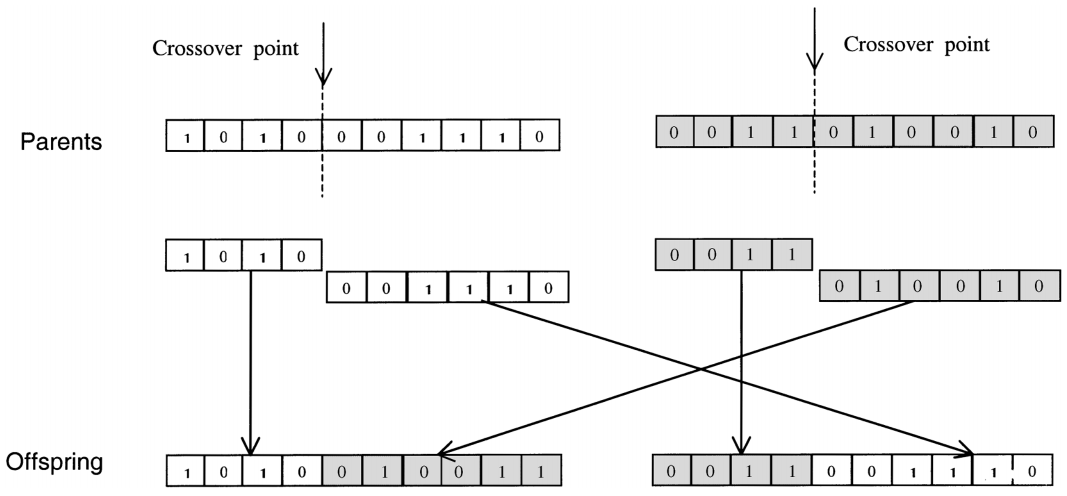


Fig. 1 Single point crossover

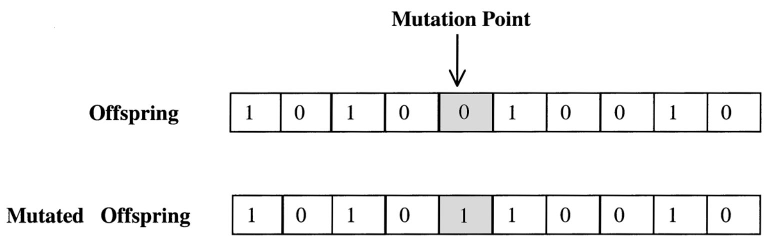


Fig. 2 Mutation

chromosomes as illustrated in Fig. 1. The two offspring can inherit some genes from each parent. The crossover operator will not be applied on all selected individuals and will be applied with a probability usually chosen by the user depending upon the application on hand.

The mutation operator is applied to each child individually after crossover. It randomly alters each gene with a small probability (typically of the range 0.001 to 0.002). Fig. 2 shows the fifth gene of the chromosome is being mutated. The mutation operator helps in exploring a search space. Mutation is traditionally seen as a “background operator” (Beasley, Bull and Martin 1993) responsible for re-introducing inadvertently ‘lost’ gene values (alleles), providing genetic drift and also provides a small element of random search in the vicinity of the population, which has largely converged.

Studies have initially been conducted on several test graphs in order to asess the effectiveness of various selection operators like roulette wheel selection method, tournament selection method and the stochastic remainder selection method. Similarly on various crossover operators like single point crossover, multi-point crossover and also uniform crossover. The studies indicated that tournament selection with two-point crossover performs better than any other combination of selection and crossover operators. Tournament selection technique has several variants. In the present work, the simplest binary tournament selection has been employed where pairs of individuals are picked at random from the population and the individual (genome) which has higher fitness is copied into the

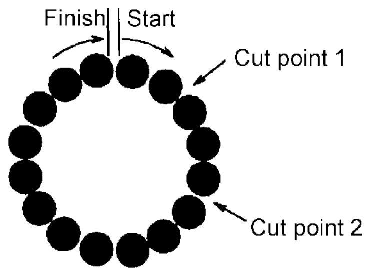


Fig. 3 Cromosome viewed as a loop for two-point crossover

mating pool. This is repeated until the mating pool is full. Similarly, in the two-point crossover employed in the present work, the strings (chromosomes) are regarded as loops formed by joining the ends together. To exchange a segment from one loop with that from another loop requires the selection of two cut points as shown in Fig. 3.

It has been observed that the convergence of the GAs is faster when applied with crossover probability of 0.85 and mutation probability of 0.0015 for the test problems and the same parameters have been employed for all the examples solved and reported later in this paper. Experiments have also been conducted on these smaller test graphs with variable mutation probability. Initially the probability is chosen as constant for the entire graph. Once majority of solutions is converged to a reasonable good bisection (say after few generations) the mutation probability is varied for each vertex according to its position in the graph. For this purpose, the interface vertices (i.e., the vertices associated with the cutedges of the bisected graph) are identified. The quality of solution is improved, when an increased mutation probability of 0.0025 is chosen for the interface vertices and 0.001 for non-interface vertices. These preliminary studies have been taken as basis for choosing appropriate parameters for GA operators. The details of these studies have been omitted, as they are of least consequence in the present context.

The cost function is computed for each of the genome (i.e., fitness) by using Eq. (1). The number of cutedges N_{ce} can be evaluated by employing the Laplacian matrix, L of the graph

$$N_{ce} = x^T L x \tag{5}$$

Where x is the vector consisting of the vertices of two partitions represented in the form of +1 and −1.

3.3 Representation of genome

Proper representation of genome in the genetic algorithms has a great influence over the performance of the solution. Hence, care must be taken to choose an appropriate representation. In K-way graph partitioning problems, the most obvious choice would be to define a genome as a list of N integers from 1 to K, each one specifying subgraph to which the corresponding vertex belongs. Similarly, in graph bisection problems, binary representation looks quite appropriate. However, this representation has disadvantages when applied to large size problems, due to the following reasons:

i. In practice, parallel processing techniques are employed only to solve large and complex problems. These problems are described by very large meshes consisting of thousands of elements. Applying GA for these problems is extremely difficult as the genome length depends on the size of the graph associated with the mesh and population size is directly related to the genome length. This will exhaust the resources of any computer and will be highly computationally inefficient. This defeats the purpose of obtaining faster and robust solutions for graph partitioning problem. Moreover, large chromosome size associated with large population size may provide solutions corresponding to local minima.

ii. If the genome is constructed with random values, vast majority of the resulting solutions is absurdly bad ones. Even if one arrives at fairly well load-balanced graph partitioning, it will result in large edge cuts making the solution worthless.

In view of this, it will be preferable to reduce the size of the genome and it is also preferable to start with some reasonably good or feasible solution (initialization of the genome). Some of the experiments made in the representation of the genome and the results are discussed in the next section.

As already discussed, the GA based graph partitioning algorithms are devised as recursive algorithms and binary encoding has been employed for employing GAs. However as already pointed out, it is impossible to solve larger problems with this representation unless the problem size is reduced. In order to reduce the problem size, two different approaches have been employed. One is by making use of the multilevel approach and the other is based on approximate bisection of the graph, followed by an interface refinement employing GA.

3.4 Multilevel approach

Multilevel approach is analogous to multi-grid methods of solution of simultaneous equations and can be effectively employed to improve the quality of the partition optimisation. This can be achieved by grouping together (clustering) small numbers of adjacent vertices to form a coarse graph. If this is done on the entire graph, it results into a coarse graph. The edge weight between the coarse vertices might then be taken as the value of the sum of the weights of all the edges

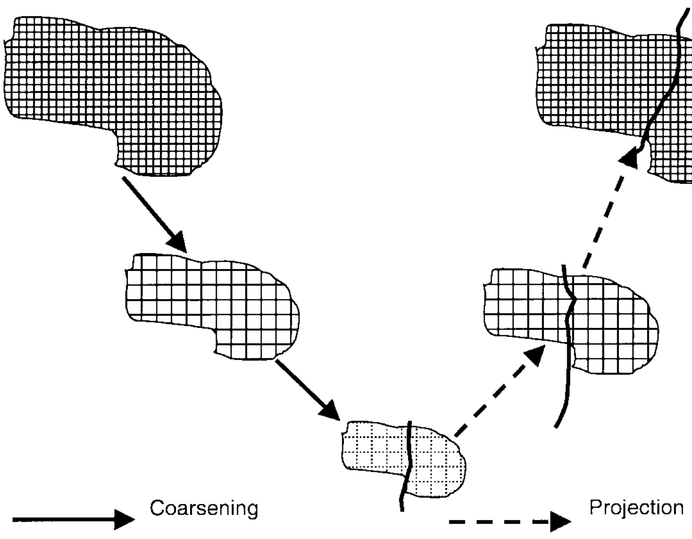
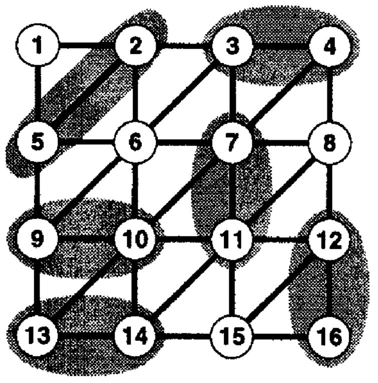
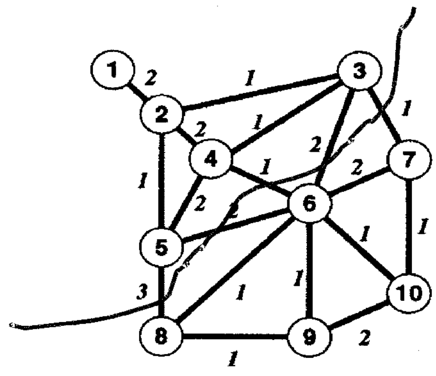


Fig. 4 Multilevel approach for graph partitioning



(a) Dual graph



Average Degree: 1.8

(b) Coarsened graph

Vertices	3	4	5	6	7	8
Binary	1	1	1	0	0	0

(c) Binary encoding of interface vertices for GA

Fig. 5 Genome representation

running on the fine graph between the two coarse graph vertices. The optimisation would then have a smaller search space and quality of the partition will generally be better than that of the finer graph. Moreover, with the reduction in size, the problem becomes manageable enough to employ GA. Once the partition of the coarser graph is obtained, it is then mapped onto the finer grid and the optimisation on the finer graph is initialised with the resulting partition.

In multilevel approach, sequence of coarsened graphs is constructed until the coarsest graph of desired size is obtained. Once the sequence of coarsened graphs has been obtained, these can be used to help in partitioning the largest graph. This is achieved by first partitioning the coarsest graph, then projecting the resulting partition onto the next graph level to initialise the optimisation on that level. The sequence of projection and refinement is carried out on all levels of coarsened graphs until the original finer graph is obtained with refined partition. This multilevel approach is schematically shown in Fig. 4.

The GA based mesh-partitioning algorithm has been developed within the framework of the multilevel approach. Here the binary encoding has been employed for applying genetic algorithms. It is based on direct mapping of each graph vertex to a bit or a single gene in a genome as shown in Fig. 5. Genes marked with 0 or 1 denote graph vertices that belong to the one or the other partition.

4. Computational procedure

The dual graph of the finite element mesh is constructed and during coarsening a sequence of smaller graphs, $G^i(V_i, E_i)$ is constructed from the original graph $G^o(V_o, E_o)$. The strategy of the graph coarsening exercise is to choose graph vertices from the graph to be coarsened, such that the coarse graph vertices are evenly distributed throughout the original graph. Their position and connectivity (represented by edges between coarse graph vertices) define a graph, which is a good coarse representation of the original graph.

For coarsening the graph, first the maximal matching of the graph is found. A matching is a subset $M \subset E$ of edges in such a way that no two edges share an end point. A matching is maximal if no other edges from E can be added. All vertices in the graph G^m are visited one by one in a random manner. All the edges connected to the vertex are sorted by their weights. The edge having the maximum weight is selected and the two corresponding vertices are collapsed and formed as a new vertex in G^{m+1}. The weight of the new vertex in G^{m+1} is equal to the sum of the weights of the

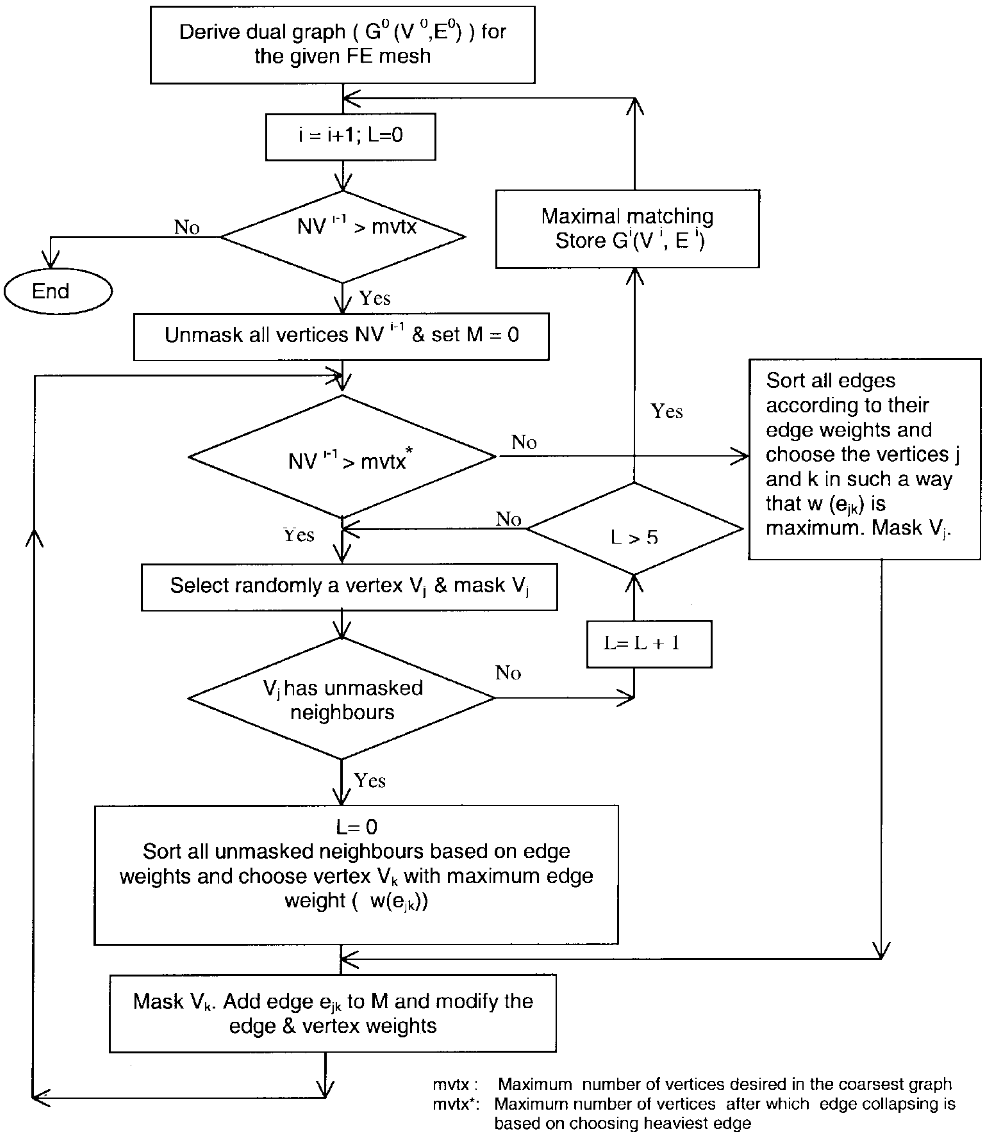

Derive dual graph (G⁰(V⁰,E⁰)) for the given FE mesh
i = i+1; L=0
NV i-1 > mvtx
No
End
Yes
Unmask all vertices NV i-1 & set M = 0
NV i-1 > mvtx*
No
Yes
Maximal matching Store Gⁱ(Vⁱ, Eⁱ)
Sort all edges according to their edge weights and choose the vertices j and k in such a way that w (ejk) is maximum. Mask Vj.
L > 5
No
Select randomly a vertex Vj & mask Vj
L= L + 1
Vj has unmasked neighbours
No
Yes
L= 0
Sort all unmasked neighbours based on edge weights and choose vertex Vk with maximum edge weight (w(ejk))
Mask Vk. Add edge ejk to M and modify the edge & vertex weights
mvtx : Maximum number of vertices desired in the coarsest graph
mvtx*: Maximum number of vertices after which edge collapsing is based on choosing heaviest edge

Fig. 6 Flow chart showing the coarsening technique employed in GA based partitioning algorithms

two G^m vertices. At the end, the remaining G^m vertices are directly upgraded to G^{m+1}, with the same weights. However, in the last few coarsening steps, the edges of the graph are sorted according to their weights and edge having the maximum weight is chosen and the corresponding vertices are collapsed. This method tends to maximise the weights of the collapsed graph edges and consequently minimise the weights of the active G^{m+1} edges. Decreasing the total edge-weight of the coarser graph, edge cut size of the coarsened graph and also the edge cut size of the original graph can be minimised. The size of G^o and the desired final coarse graph size determine the total number 'm' of the coarsening steps. For employing GA, the size of coarsest graph has been limited to 50 vertices. The graph coarsening procedure is shown in the form of a flowchart in Fig. 6.

The second phase of a multilevel algorithm is to compute balanced bisection of the coarsest graph $G^k(V_k, E_k)$. For this purpose, two different alternatives have been attempted. One alternative is to employ any one of the cost-effective heuristic mesh-partitioning algorithms for bisecting the coarsest graph. The other alternative is to bisect the graph employing genetic algorithms. The first alternative has been termed as MLGA-1 (Multi-Level Genetic Algorithm-1) and greedy algorithm (Farhat 1988) has been employed for bisecting the coarsened graph. However, greedy algorithm is known to be sensitive to the chosen seed vertex (Rama Mohan Rao 2001). In view of this, greedy algorithm is repetitively employed on the coarsest graph choosing each time a different seed vertex and the best partition (i.e., with minimum edge-cuts) among them is chosen. In the second alternative, MLGA-2, the coarsest graph is bisected using genetic algorithm.

In the proposed GA implementation, tournament selection scheme has been employed with two-point crossover. A customized mutation operator has been employed during bisection of the coarsest graph. Contrary to the conventional mutation operator, the customized operator changes bits either from zero to one or vice versa in a particular string. This will be decided randomly on each string. If the initial population is generated randomly, this one way mutation helps in improving the load balancing in some strings and may cause damage to others. The damaged ones are generally eliminated in selection. If they are not eliminated, they can recover in the subsequent generation.

As already pointed out earlier, if the population is initialised with random values of 0 and 1, then the fitness of the population is expected to be very low. In view of this, some sort of an operator which can quickly improve fitness of this population will ultimately help in accelerating the convergence characteristics of GA. For example, if a particular gene value is zero in a typical string and all it's neighbours have value one, then it is preferable to swap this gene value to one. In other words if a particular element in a mesh is marked to the first submesh (with corresponding gene value as 0) and its surrounding elements (with gene value as 1) belonging to the other submesh, it is preferable to change the ownership of this typical element. This is achieved by the gene-repair strategy employed in the present work. This will be operated only in the first few generations. In order to minimise the overheads associated with this operator, this gene repair strategy is applied only at every 3rd generation. As already explained, its objective is to change the value of isolated genes. The gene repairing strategy is employed after the mutation and before computing the fitness. In every 3rd generation, 5% of the strings are randomly picked from the population. Each string is scanned through for isolated genes and their values will be altered to the values of the surrounding ones. Once the quality of the partitions is improved after few generations, the gene repair strategy will not be employed.

However, after obtaining a reasonably good bisection of the graph, mutation with variable probability has been employed. The mutation takes into consideration the graph connectivity. Higher probability (0.0025) is given to the interface vertices and lower (starting from 0.001 and adaptively

```
Algorithm: Balance
1. Identify the direction of swapping based on the size of the two projected subgraphs.
   Mark it as active
2. Collect a pool of vertices from the active subgraph in such a way that they are either
   interface vertices or vertices of a disjointed part of the subgraph or others in the same
   order of preference.
3. Unmask all vertices in the pool
4. Set the balance parameter as the difference in sizes of vertices of the two subgraphs
5. While (balance parameter >1), DO
  i. Extract an unmasked vertex from the pool of the active subgraph list
 ii. Check if edge cuts can be reduced by swapping the vertex
      If, yes
           (a) Move the vertex to the inactive subgraph
           (b) Update the interface vertex data structure
           (c) Update the balance parameter
      Else
            Mask the vertex in the pool
     Endif
ENDDO
6. Update the vertex list of the active subgraph
7. if(balance parameter>1)
    i. Unmask all vertices
   ii. Collect pool of vertices as in Step-2 and go to step-5
End Algorithm
```

Fig. 7 Heuristic balancing algorithm

reduces to zero as the solution converges) probability is given to the internal vertices.

The third phase of the proposed algorithm is uncoarsening. During uncoarsening phase, the partition of the coarsest graph G^k is projected back to the original graph by going through the graphs G^{i-1}, G^{i-2},.... G^i. Since each vertex $u \in V^i$ contains a distinct sub set U of vertices V^{i-1}, projecting the partition of G^i to G^{i-1} is done by simply assigning the vertices in U to the same part that vertex u belongs to. Even if the original coarsest graph is in local minima, the projected graph may not be at local minima. Since the projected graph is much finer and has more degrees of freedom, these vertices can be used for improving the quality of partition. Once the graph is projected to the next finer level, the balancing is performed using the new heuristic-balancing algorithm given in Fig. 7. The balancing algorithm is terminated as soon as the partitions are balanced.

After balancing, the genetic algorithm is employed for improving the quality of partitions. However, the genetic algorithm can only be applied by considering only limited number of vertices in the graph. Otherwise the problem becomes quite unmanageable. In view of this, only the interface layers of the projected graph are considered and attempts have been made to improve the partitions employing the genetic algorithm. The genetic algorithms are employed at each alternative projection in order to minimise the computational cost. The maximum number of vertices for genetic algorithms will be of the order of 50 to 80 depending on the size of the problem. If the population in GA is initialised with random values of 1 and 0, the schema will be generally of poor quality and convergence will be rather slow. In view of this, at least 50% of the population are initialised with the reasonably good partitioned information available at the projected level after balancing (elitism). This certainly helps in accelerating the convergence. The sequence of projection,

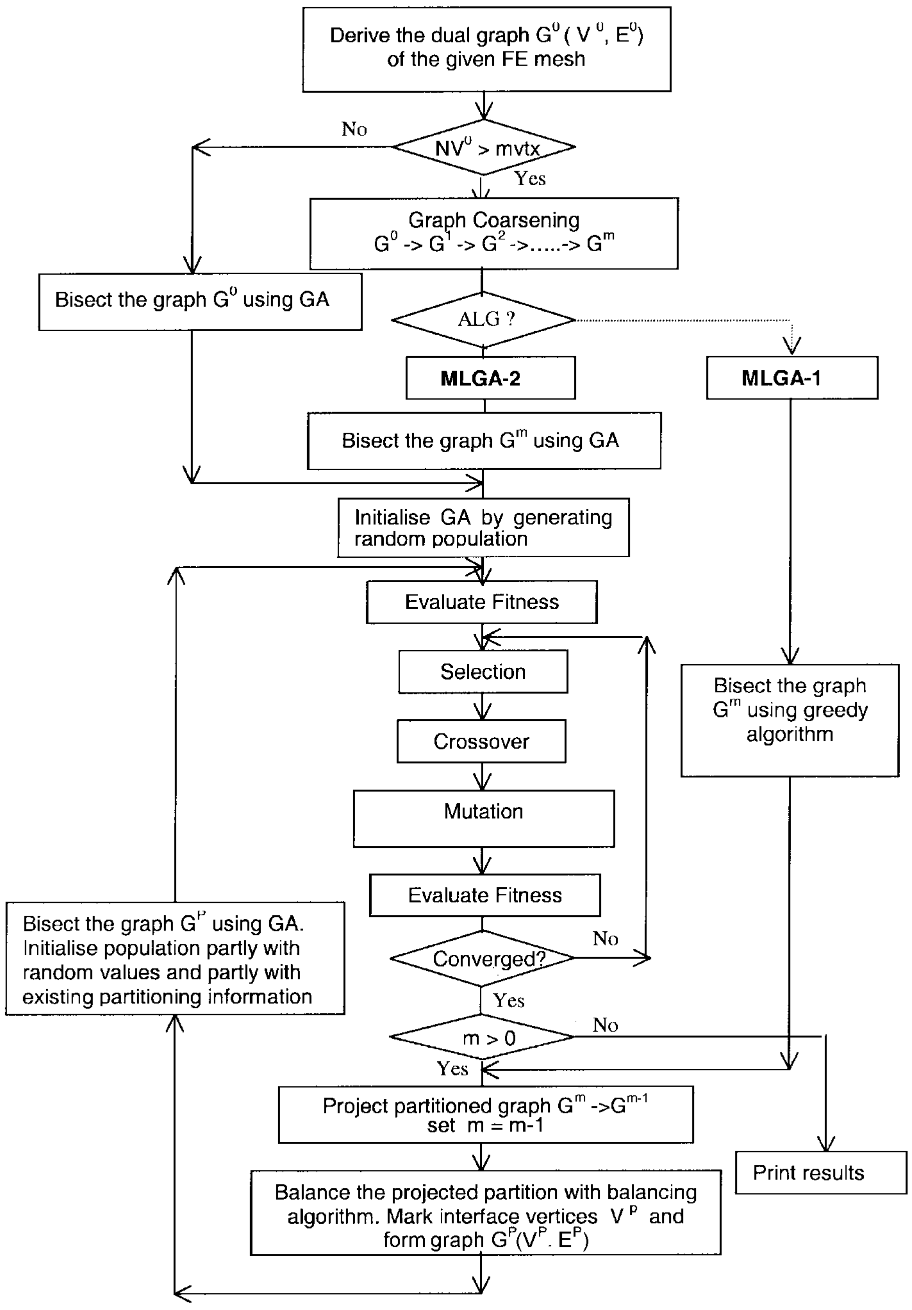

Derive the dual graph $G^0(V^0, E^0)$ of the given FE mesh
No
NV^0 > mvtx
Yes
Graph Coarsening $G^0 \to G^1 \to G^2 \to \ldots \to G^m$
Bisect the graph G^0 using GA
ALG ?
MLGA-2
MLGA-1
Bisect the graph G^m using GA
Initialise GA by generating random population
Evaluate Fitness
Selection
Crossover
Mutation
Evaluate Fitness
Converged?
No
Yes
Bisect the graph G^m using greedy algorithm
Bisect the graph G^P using GA. Initialise population partly with random values and partly with existing partitioning information
m > 0
No
Yes
Project partitioned graph $G^m \to G^{m-1}$ set m = m-1
Print results
Balance the projected partition with balancing algorithm. Mark interface vertices V^P and form graph $G^P(V^P, E^P)$

Fig. 8 Flow chart of GA based mesh partitioning algorithms

balancing and GA refinement is carried out until the coarser graph is projected back to the original graph G^0 and refined using genetic algorithms. The implementation of MLGA-1 and MLGA-2 is shown in the flow chart given in Fig. 8.

During refinement, the adaptive mutation based on the graph connectivity has been employed. The mutation probability has been fixed to each vertex according to its position. Customized mutation has been implemented by using a mutation factor. The bias for each bit in the string is based on the mutation factor. This mutation factor can be defined as the gain of a vertex (in terms of reduction in number of cut edges) if it is swapped to the next partition. This customized mutation contributes in swapping the bits (corresponds to the vertices with high gain) and thereby enhance the quality of partition.

5. Local Refinement With GA (LRGA)

In this approach, the graph coarsening procedure is dispensed with. Instead, the graph associated with the finite element mesh is bisected using any of the fast heuristic partitioning algorithm and the interfaces are identified. In the present implementation, greedy algorithm (Farhat 1988) has been employed. A few strips surrounding the interface vertices are considered and GA is employed on this reduced Graph. However, objective function is evaluated using the total graph in order to avoid local partitioning. The computation of objective function is faster as it involves only in identifying the interfaces, N_{ce} and the vertices within the strip of the graph belonging to different partitions. This avoids the process of identifying all the vertices belonging to different groups in the entire graph, which in fact includes many vertices that are not expected to change the sides. In this approach, only a single strip of the graph surrounding the approximately identified interface vertices are considered in order to bring down the size to a manageable level on which GAs can be employed efficiently.

6. Evaluation of partitioning algorithms

The proposed GA based mesh partitioning algorithms have been integrated into PSTRAIN (Parallel STructural Analysis INterface) software, which consists of variety of mesh partitioning algorithms (Rama Mohan Rao *et al.* 1998). PSTRAIN is built with powerful Graphic User Interface using X, Motif and OpenGL in order to facilitate visualisation.

In order to demonstrate the effectiveness of the proposed mesh partitioning algorithms, several unstructured meshes of practical engineering problems have been considered as a first set of numerical examples. In this paper four of such numerical examples are presented. For the numerical examples discussed in this paper, population size is taken as 50. Two-point crossover has been employed with probability as 0.85 and mutation with probability as 0.0015. Customized adaptive mutation as discussed in the earlier sections has been employed. As already discussed earlier, the GA is employed for refining the alternative projected graphs solely to cut down the computational cost. The solution is assumed to have converged if eight consecutive generations fail to improve the solution.

The first numerical example considered is an unstructured mesh of a typical joint (JOINT1) of an off-shore platform. The number of vertices in the corresponding dual graph are 1182 and edges are 957. The mesh is partitioned into 2^n partitions where n is varying from 1 to 5. The cut edges and also the CPU time for generating the partitions is given in Table 1. It can be seen from the results that the GA based algorithms generate minimum cut edges. However, the computational cost is considerably high. The local refinement algorithm (LRGA) appears to be faster than the multilevel

Table 1 Performance of GA based partitioning algorithms for JOINT1

NP	MLGA-1		MLGA-2		LRGA		MRSB
	EC	CPU time in Sec.	EC	CPU time in Sec.	EC	CPU time in Sec.	EC
2	68	28.48	66	42.3	88	15.86	102
4	178	48.88	172	89.91	192	27.92	189
8	319	98.40	314	139.56	342	48.31	338
16	482	216.94	486	264.78	502	73.89	495
32	676	444.61	699	509.08	716	103.43	696

NP: Number of partitions, EC: Cut edges

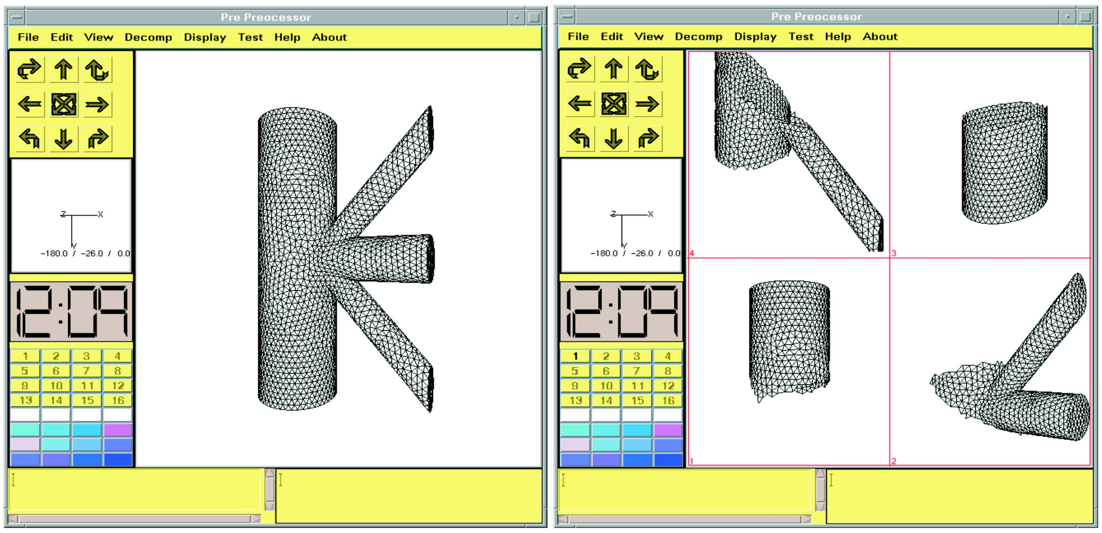


Fig. 9 Generation of four partitions of an unstructured mesh describing a typical joint of an off-shore platform employing MLGA-2

Table 2 Performance of GA based partitioning algorithms for JOINT2

NP	CUT EDGES (EC)			
	MLGA-1	MLGA-2	LRGA	MRSB
2	90	90	122	92
4	196	187	214	192
8	319	316	347	326
16	527	522	588	516
32	812	804	916	814

NP: Number of partitions

algorithm with rather larger cut edges. Fig. 9 show the partitioning results of the unstructured mesh employing MLGA-2.

Similarly, the second numerical example considered is a joint (JOINT2) of a space frame

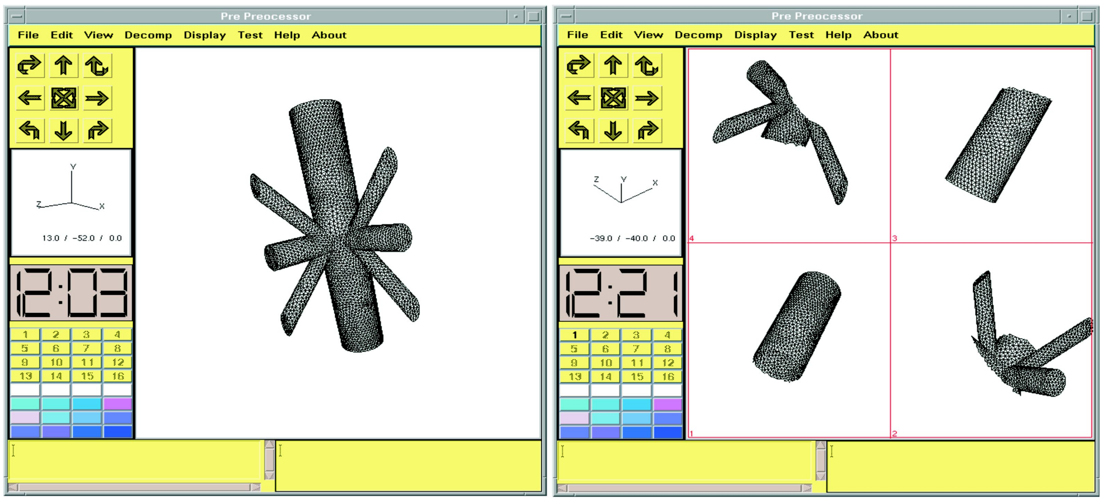


Fig. 10 Generation of four partitions of an unstructured mesh describing a typical joint of a space frame employing MLGA-2

Table 3 Performance of GA based partitioning algorithms for 3D FEMESH

NP	CUT EDGES(EC)			
	MLGA-1	MLGA-2	LRGA	MRSB
2	371	368	392	-
4	2789	2778	3301	3292
8	6576	6531	6946	6895
16	9979	9967	10486	10397
32	13221	13164	14489	14499
64	18896	18826	19823	19772

NP: Number of partitions

described by an unstructured mesh with triangular elements. The number of vertices in the corresponding dual graph is 12900 and edges are 18880. The partitioning results of the tube are presented in Table 2. In this example, also the cut edges obtained by LRGA are high when compared to multilevel implementation. The partitioned mesh is shown in Fig. 10.

The third numerical example considered is a thick metal sheet described by a graded three-dimensional mesh. It consists of 19334 brick elements. The corresponding dual graph has 19334 vertices and 55305 edges. The partitioning results are shown in Table 3. It can be seen from the results that the number of cut edges obtained by MLGA-1 and MLGA-2 are more or less same. However, the MLGA-1 is slightly faster than MLGA-2 as GA is not employed in the bisection of the coarsened graph. Instead, a faster heuristic algorithm has been employed. The numerical studies indicate that GA based algorithms outperform the Multi-level spectral algorithms (MRSB) (Barnard and Simon 1994) in terms of minimising the cut edges. However, the computational cost of these GA based algorithms is much higher than the multilevel spectral approaches.

The fourth numerical example considered is body of an automobile (AUTO-B) described by an

Table 4 Performance of GA based partitioning algorithms for mesh describing body of an automobile

NP	CUT EDGES(EC)		
	MLGA-1	MLGA-2	MRSB
4	20684	20589	33214
8	32432	31887	61326
16	54947	54855	82127
32	80522	80631	128369
64	126832	126653	193291

NP: Number of partitions

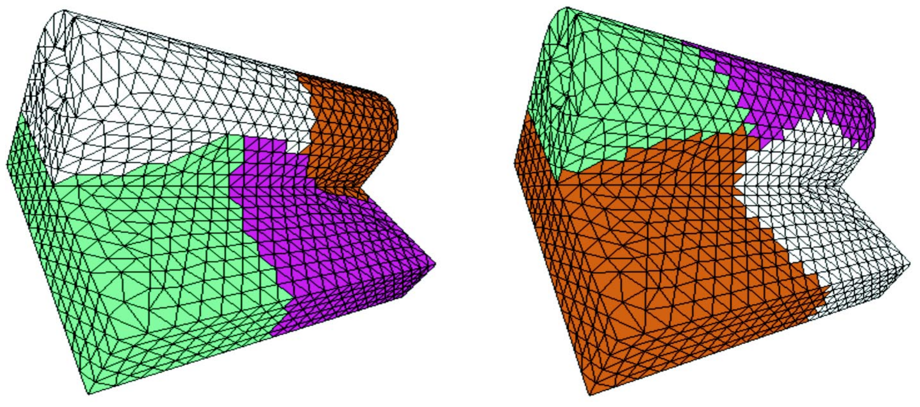

(a) Recursive Spectral Bisection algorithm
(b) Genetic algorithm (MLGA-2)

Fig. 11 Generation of four submeshes of an unstructured mesh employing RSB and MLGA-2

unstructured finite element mesh. The vertices in the associated dual graph are 327468 and the edges are 9557804. Table 4 shows the results obtained with spectral algorithm and GA based partitioning algorithm. A close look at the results indicate that, the GA based partitioning algorithms are superior to spectral approaches in terms of minimisation of cutedges.

Fig. 11 shows the partitions generated by employing recursive spectral algorithm and also MLGA-2 of a mesh describing a mechanical bracket. Dual graph of the mesh is used for partitioning. It can be observed that the partitions generated by both these algorithms are distinct. The number of cut edges of the associated partitioned graphs is 179 and 162 respectively for RSB and MLGA-2.

The next set of numerical experiments have been conducted by solving some of the test graphs of finite element meshes (both 2D and 3D) available in the literature and accessible through net. The details of these test graphs are shown in Table 5. The results of the proposed GA based graph partitioning algorithm have been compared with the multilevel graph partitioning algorithm METIS. For this purpose, METIS Ver 4.0 is implemented and results are obtained by employing both *pmetis* and *kmetis*. The comparison between METIS and the proposed GA based graph partitioning algorithm is shown in Table 6. MLGA-2 is used to generate the partitions. The results shown in the Table 6 are the total number of cutedges generated by the partitioning algorithms. It is appropriate

Table 5 Details of test graphs

SNO	Graph Name	Number of Vertices	Number of Edges	Description
1.	3ELT	4720	13722	3D finite element mesh
2.	4ELT	15606	45878	3D finite element mesh
3.	CRACK	10240	30380	3D finite element mesh
4.	CTI	16840	48232	3D finite element mesh
5.	FE_4ELT2	11143	32818	3D finite element mesh
6.	FE_BODY	45087	163734	3D finite element mesh
7.	FE_SPHERE	16386	49152	3D finite element mesh
8.	T60K	60005	89440	2D finite element mesh
9.	UK	4824	6837	2D finite element mesh
10.	WHITAKER3	9800	28989	3D finite element mesh
11.	WING	62032	121544	3D finite element mesh
12.	WING-NODAL	10937	75488	3D finite element mesh
13.	BRACK2	62631	366559	3D finite element mesh
14.	FE_ROTOR	99617	662431	3D finite element mesh
15.	FE_TOOTH	78136	452591	3D finite element mesh
16.	AUTO	448695	3314611	3D finite element mesh
17.	WAVE	156317	1059331	3D finite element mesh
18.	144	144649	1074393	3D finite element mesh
19.	598a	110971	741934	3D finite element mesh
20.	CS4	22499	43858	3D finite element mesh

Table 6 Comparison of cut-edge results of MLGA-2 and METIS

SNO	Graph	Algorithm	NP = 2	NP = 4	NP = 8	NP = 16	NP = 32	NP = 64	NP = 128
1.	3ELT	MLGA-2	98	201	344	608	1006	1585	2480
		PMETIS	110	215	388	684	1087	1726	2628
		KMETIS	109	257	376	645	1066	1662	2571
2.	4ELT	MLGA-2	128	324	554	998	1692	2746	4408
		PMETIS	142	368	602	1072	1804	2861	4628
		KMETIS	149	359	744	1148	1823	2878	4426
3.	CRACK	MLGA-2	187	434	736	1187	1864	2784	3966
		PMETIS	209	476	804	1269	1961	2899	4277
		KMETIS	219	478	784	1310	1939	2846	4063
4.	CTI	MLGA-2	311	973	1868	2941	4487	6347	8419
		PMETIS	371	1030	1962	3079	4661	6522	8552
		KMETIS	330	1098	2201	3479	4667	6627	8852
5.	FE_4ELT2	MLGA-2	112	312	594	1046	1643	2652	3936
		PMETIS	130	359	663	1140	1800	2809	4151
		KMETIS	130	391	657	1127	1753	2744	4022
6.	FE_BODY	MLGA-2	237	592	1064	1965	3388	5576	9218
		PMETIS	283	689	1220	2109	3566	5758	9434
		KMETIS	292	843	1224	2285	3513	7992	10190
7.	FE_SPHERE	MLGA-2	382	766	1182	1832	2744	3885	5527
		PMETIS	424	843	1328	2031	2964	4200	5921
		KMETIS	446	847	1318	1982	2867	4042	5679

Table 6 Comparison of cut-edge results of MLGA-2 and METIS (continued)

SNO	Graph	Algorithm	NP = 2	NP = 4	NP = 8	NP = 16	NP = 32	NP = 64	NP = 128
8.	T60K	MLGA-2	88	228	506	918	1536	2408	3552
		PMETIS	102	266	519	1012	1629	2503	3751
		KMETIS	102	307	573	1006	1638	2501	3684
9.	UK	MLGA-2	27	46	104	169	297	438	727
		PMETIS	27	53	115	199	326	491	778
		KMETIS	31	52	116	195	305	477	778
10.	WHITAKER3	MLGA-2	128	398	698	1164	1768	2636	3824
		PMETIS	131	429	716	1216	1839	2817	4116
		KMETIS	135	431	746	1198	1832	2676	3962
11.	WING	MLGA-2	742	1883	2806	4366	6465	8862	11627
		PMETIS	882	1986	2912	4634	6717	9153	12387
		KMETIS	929	1997	2984	4445	6761	9029	11937
12.	WING-NODAL	MLGA-2	1788	3682	5994	8964	12014	16061	21689
		PMETIS	1813	4033	6256	9346	13333	18003	23286
		KMETIS	1927	3801	6185	9087	12832	17051	22039
13.	BRACK2	MLGA-2	666	3016	7543	12369	19234	28884	40876
		PMETIS	742	3426	8000	13358	19524	29481	41485
		KMETIS	742	3216	7960	13001	19976	29350	42429
14.	FE_ROTOR	MLGA-2	2052	8010	14022	24121	35214	51680	73854
		PMETIS	2647	8354	15359	24841	35993	52309	75250
		KMETIS	2237	8430	14733	25128	38131	53601	74082
15.	FE_TOOTH	MLGA-2	4298	7767	13092	19642	27911	38396	52596
		PMETIS	4618	7854	13970	20682	28952	40170	54549
		KMETIS	4382	9002	13640	20069	28715	39160	53290
16.	AUTO	MLGA-2	8604	28412	46864	86019	127981	185852	254903
		PMETIS	9336	30240	53326	90291	138008	197461	270117
		KMETIS	10924	31856	49733	89731	130086	188087	259264
17.	WAVE	MLGA-2	8812	19212	32166	46984	67156	91096	123694
		PMETIS	9336	21349	35835	52243	71419	96445	130043
		KMETIS	9355	20168	33553	48594	69501	93195	125100
18.	144	MLGA-2	6212	16328	26586	41124	61011	84026	114007
		PMETIS	6919	17647	29270	43253	63576	89309	121319
		KMETIS	6996	17647	28416	43172	63415	86262	116471
19.	598a	MLGA-2	2208	8133	17656	29586	43164	61644	84412
		PMETIS	2444	8631	18328	30170	44555	64150	88658
		KMETIS	2555	9003	18133	30274	43877	62617	85830
20.	CS4	MLGA-2	388	1022	1643	2387	3187	4608	5916
		PMETIS	421	1137	1732	2540	3527	4776	6383
		KMETIS	494	1102	1759	2464	3404	4724	6164

to point out here that while generating the partitions employing the proposed algorithm for these test meshes, greater emphasis is placed on quality rather than on computational time. Hence the computational time is not measured for these problems. In order to improve the quality, the GA is employed for refining the projected graph partitions at each level. The GA is terminated when there is no improvement in the solution for twenty consecutive generations. A close look at Table 5 indicates that the cutedges generated by the proposed GA based graph partitioning algorithm are lower than the multilevel algorithm.

Comparisons have also been made with another graph partitioning package CHACO. Since CHACO code is not accessible, the values available in the literature (Soper, Walshaw and Cross 2000) for CHACO version 2.0 are taken for comparison purposes and the results are furnished in Table 7. A close look at these results also indicates the superior performance of the proposed GA based graph partitioning algorithm. Similar comparisons have been made with JOSTLE 2.2. It is again preferred to borrow results (cut edges) of JOSTLE from the literature (Soper, Walshaw and Cross 2000) rather than implementing and testing on our own. The comparison of cutedges is shown in Table 8. It can be observed that the proposed algorithm is effective in minimising the cut edges in most of the situations. Only in very few instances, JOSTLE is found to be marginally better.

Table 7 Comparison of cut-edge results of MLGA-2 and CHACO

Graph	NP = 4		NP = 8		NP = 16		NP = 32	
	MLGA-2	CHACO	MLGA-2	CHACO	MLGA-2	CHACO	MLGA-2	CHACO
CRACK	434	466	736	794	1187	1246	1864	1978
CTI	973	1006	1868	1946	2941	3091	4487	4583
T60K	228	244	506	526	918	994	1536	1601
UK	46	63	104	121	169	202	297	318
WING	1883	2104	2806	3409	4366	4880	6465	6800
WING-NODAL	3682	3979	5994	6145	8964	9353	12014	13272
BRACK2	3016	4116	7543	9183	12369	14683	19234	22610
4ELT	324	384	554	682	998	1155	1692	1745

Table 8 Comparison of cut-edge results of MLGA-2 and JOSTLE

Graph	NP = 4		NP = 8		NP = 16		NP = 32	
	MLGA-2	JOSTLE	MLGA-2	JOSTLE	MLGA-2	JOSTLE	MLGA-2	JOSTLE
CRACK	434	413	736	751	1187	1191	1864	1804
CTI	973	1329	1868	2086	2941	3262	4487	4683
T60K	228	229	506	530	918	984	1536	1588
UK	46	71	104	106	169	180	297	315
WING	1883	1844	2806	2911	4366	4681	6465	6404
WING-NODAL	3682	4055	5994	5965	8964	8947	12014	12635
BRACK2	3016	2999	7543	7808	12369	13164	19234	19238
4ELT	324	434	554	656	998	1012	1692	1687

7. Conclusions

In this paper, an unstructured mesh-partitioning technique employing genetic algorithms has been presented. The algorithms discussed in this paper have been devised as recursive algorithms. In order to formulate GA based mesh-partitioning algorithms, the genome is constructed by using binary representation and by mapping the graph vertices to the bit string. Both direct as well as multilevel methods have been devised employing this formulation. It has been observed that customized mutation operator and elitism contributes for faster convergence of GA in multilevel implementation. GA based partitioning algorithm (MLGA-1) is slightly faster if coarsest graph is bisected using any of the fast heuristic graph partitioning algorithms

An alternative approach is also presented, which utilizes the greedy algorithm for approximate bisection of the mesh and GA is applied on the vertices along the interfaces. This not only reduces the size of genome but also have an effect on the computational time, as the cost function evaluation is limited to only the negotiable vertices, i.e., the interface vertices. In other words, GA works in this particular context as a local refinement algorithm. However, the cut edges obtained using this formulation are inferior to the multilevel approach.

Numerical studies employing various practical engineering problems indicate that the proposed GA based algorithms outperform the popular spectral approaches in terms of minimising the cutedges. Among the three different implementations, the MLGA-2 is qualitatively superior. However, the computational cost is high. Similarly several test graphs of various finite elements meshes have also been solved employing MLGA-2. The cut edges generated by the proposed graph-partitioning algorithm are much lower than several state-of-the-art mesh partitioning algorithms like METIS, CHACO.

In this work, the finite element meshes with mixed element are not tested. However, it is worth mentioning here that the finite element meshes with elements of varying order and also meshes with mixed finite elements, (which result in weighted graphs) can be solved employing the proposed algorithm without any difficulty.

In the light of the numerical studies and the comparisons with various popular mesh partitioning algorithms, the merits of the proposed GA based graph partitioning algorithm and their potential applications can be compiled as given below:

1. The GA based partitioning algorithm is robust and guarantees to provide a solution. The quality of solution improves with time.
2. The proposed GA based graph partitioning algorithm produces high quality meshes. Hence they are recommended for finite element applications of large models typically say for nonlinear static or dynamic analysis (where incremental and/or iterative form of solutions are usually employed) whose runtime on parallel machines will usually be high. For example, the crash worthiness studies will generally be carried out for about 100,000 time steps (Plaskacz *et al.* 1994) and the run time of this sort of applications will be about few hours. In such situations, superiority in the quality of partitions is preferred at the expense of slightly higher run time in generating the partitions. In other words, if one can minimise the cut edges further even at the expense of an additional computing time, considerable savings can be expected during finite element solution as these applications(nonlinear/nonlinear dynamics) require interprocessor communication in each time step/iteration or increment. Similarly the proposed algorithm is effective for computational fluid dynamics (CFD) applications.

3. The proposed algorithm attempts to optimise the cost function associated with graph partitioning. Hence the user has flexibility to tune the cost function to his specific requirements or improve the cost function by introducing additional partitioning information like aspect ratio optimisation etc. The same code can still be used with this improved/tuned cost function. This flexibility is lacking in most of the state-of-the-art graph partitioning algorithms.
4. Since GA is an excellent candidate for parallel processing, the proposed GA based graph-partitioning algorithm is more adaptive for parallel implementation than the multilevel algorithms.
5. The major disadvantage of the proposed algorithm is the high run time. In view of this, the proposed algorithm is not recommended for problems whose run time on parallel processors is very small. For example, problems like parallel sparse solvers are expected to take only few seconds. For these class problems, fast multilevel graph partitioning algorithms like METIS, CHACO, JOSTLE are still preferable when compared to the proposed algorithms.

Acknowledgements

This paper is being published with the permission of Director, Structural Engineering Research Centre, Madras.

References

Altman, E.R., Agarwal, V.K. and Rao, G.R. (1993), "A novel methodology using genetic algorithms for the design of caches and cache replacement policy", *Proc. Int. Conf. on Genetic Algorithms*, 392-399.

Barnard, S.T. and Simon, H.D. (1994), "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems", *Concurrency: Practice and Experience*, **6**, 101-107.

Beasley, D., Bull, D.R. and Martin, R.R. (1993), "An overview of genetic algorithms: part 2, research topics", *University Computing*, **15**(4), 170-181.

Bouhmala, N. and Pahud, M. (1998), "A parallel variant of simulated annealing for optimizing mesh partitions on workstations", *Advances in Engineering Software*, **29**(3-6), 481-485.

De Keyser, J. and Roose, D. (1992), "Grid partitioning by inertial recursive bisection", Technical Report TW 174, Department of Computer Science, Belgium.

Farhat, C. (1988), "A fast simple and efficient automatic FEM domain decomposer", *Computers and Structures*, **28**(5), 579-602.

Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A guide to the theory of NP-Completeness*, Freeman, W.H. and Company, N.Y.

Gil, C., Ortega, J., Diaz, A.F. and Monotoya, M.G. (1998), "Annealing based heuristics and genetic algorithms for circuit partitioning in parallel test generation", *Future Generation Computing Systems*, **14**(5), 439-451.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning Addison*, Wesley, N.J.

Goldberg, D. and Segrest, P. (1987), "Finite Markov chain analysis of genetic algorithms", *Proc. the Fifth Int. Conf. on Genetic Algorithms*.

Hendrickson, B. and Leland, R. (1995), "A multilevel algorithm for partitioning graphs", *Proc. the Supercomputing 95*, ACM.

Holland, J. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press. Ann Arbor. MI.

Jones, M.T. and Plassmann, P.E. (1994), "Computational results for parallel unstructured mesh computations", *Computing Systems in Engineering*, **5**(4-6), 297-309.

Karypis, G. and Vipin Kumar. (1999), "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM Journal of Scientific Computing*, **20**(1), 359-392.

Kernighan, B.W. and Lin, S. (1970), "An efficient heuristic procedure for partitioning graphs", *The Bell System Technical Journal*, **29**(2), 291-307.

Khan, A.I. and Topping, B.H.V. (1998), "Subdomain generation for parallel finite element analysis", *Computing Systems in Engineering*, **4**(4-6), 96-129.

Mansoor, N. and Fox, G.C. (1991), "A hybrid genetic algorithm for task allocation in multi-computers", *Proc. the 4th Int. Conf. on Genetic Algorithms* (Ed. Belew, R.K. and Booker, L.B.), Morgan Kaufmann, 466-473.

Mansour, N. and Fox, G.C. (1994), "Allocating data to distributed memory multiprocessors by genetic algorithms", *Concurrency: Practice and Experience*, **6**(6), 485-504.

Miller, G.L., Teng, S., Thurston, W. and Vavasis, S.A. (1998), "Geometric separators for finite element meshes", *SIAM Journal of Scientific Computing*, **19**(2), 364-386.

Mitchel, W.J., Steadman, J.P. and Liggett, R.S. (1976), "Synthesis and optimisation of small rectangular floor plans", *Environment and Planning*, **3**, 37-70.

Pain, C.C., De Oliveira, C.R.E. and Goddard, A.J.H. (1999), "A neural network graph partitioning procedure for grid based domain decomposition", *Int. J. Numer. Methods Eng.*, **44**, 593-613.

Park, K. and Carter, B. (1995), "On the effectiveness of genetic search in combinatorial optimisation", *Proc. the 10th ACM Symposium on Applied Computing*, Genetic Algorithms and Optimisation Track.

Plaskacz, E.J., Ramirez, M.R. and Gupta, S. (1994), "Non-linear explicit transient finite element analysis on the intel delta", *Computing Systems in Engineering*, **5**, 1-17.

Punch, W., Goodman, E., Pei, M., Lai, C.S., Hovland, P. and Enbody, R. (1993), "Intelligent clustering of high dimensionality data using genetic algorithms", *Proc. Int. Conf. on Genetic Algorithms*, 557-564 .

Rama Mohan Rao, A. (2001), "Efficient parallel processing algorithms for nonlinear dynamic analysis", Ph.D. Thesis Submitted to Indian Institute of Science, Bangalore. India.

Rama Mohan Rao, A., Appa Rao, T.V.S.R. and Dattaguru, B. (1998), "Load balancing for parallel finite element analysis employing artificial neural networks", In CDROM *Proc. Int. Conf. on Theoretical Applied Computational and Experimental Mechanics*.

Rama Mohan Rao, A., Umesha, P.K. and Loganathan, K. (1998), "PSTRAIN: A graphic user interface for parallel nonlinear dynamic analysis of structures", SERC Technical Report NO. GAP 0641-PPG-TR-98-02.

Simon, H.D. (1991), "Partitioning of unstructured problems for parallel processing", *Computing Systems in Engineering*, **2**, 135-148.

Soper, A.J., Walshaw, C. and Cross, M. (2000), "A combined evolutionary search and multilevel optimisation approach to graph partitioning", Mathematics Research Report 00/IM/58.

Walshaw, C. and Cross, M. (1999), "Parallel optimisation algorithms for multilevel mesh partitioning", Mathematics Research Report 99/IM/44.

Wendl, S. (1996), "A seed based decomposition algorithm employing genetic algorithms", Report No. EPCC-SS96-01, University of Edinburgh, UK.

Williams, R.D. (1991), "Performance of dynamic load balancing algorithms for unstructured mesh calculations", *Concurrency*, **3**, 457-481.