Generating optimised partitions for parallel finite element computations employing float-encoded genetic algorithms

A. Rama Mohan Rao[1], **T.V.S.R. Appa Rao**[2] **and B. Dattaguru**[3]

Abstract: This paper presents an algorithm for automatic partitioning of unstructured meshes for parallel finite element computations employing float-encoded genetic algorithms (FEGA). The problem of mesh partitioning is represented in such a way that the number of variables considered in the genome (chromosome) construction is constant irrespective of the size of the problem. In order to accelerate the computational process, several acceleration techniques like constraining the search space, local improvement after initial global partitioning have been attempted. Finally, micro float-encoded genetic algorithms have been developed to accelerate the computational process.

Numerical experiments have been conducted to demonstrate the effectiveness of the GA based partitioning algorithms. The proposed algorithms have been tested on unstructured meshes describing practical engineering problems. Apart from these meshes, several benchmark problems available in the literature are also considered to evaluate the performance of the GA based partitioning algorithms. Results indicate that the proposed algorithms are qualitatively superior to popular spectral approaches and multilevel algorithms. It was also shown through numerical experiments that the micro-float-encoded genetic algorithms provide faster solutions with slight reduction in the quality when compared to float-encoded genetic algorithms.

keyword: Parallel computing, unstructured meshes, Float- encoded genetic algorithm, Micro-genetic algorithm, dual graph

[1] Scientist, Structural Engineering Research Centre, Chennai, India
[2] Former Director, Structural Engineering Research Centre, Chennai, India
[3] Chairman & Professor, Dept. Aerospace Engg., IISc, Bangalore, India

1 Introduction

Domain decomposition methods are widely used for parallel finite element computations. The necessary first step of the domain decomposition based parallel algorithms is partitioning of the given finite element mesh into specified number of submeshes which is usually equal to the number of processors. Distributing the mesh across a parallel computer so that the computational load is evenly balanced and the data locality is maximised is usually referred to as mesh partitioning. The quality of the mesh partitioning can seriously effect computing time of parallel finite element analysis. Hence greater emphasis is required to the problem of mesh partitioning while developing parallel algorithms for finite element computations.

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,... ..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.

In recent years, much attention has been focussed on developing suitable heuristics, and some powerful methods have been developed based on graph theory. 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, Teng, Hurston and Vavasis (1998)] such as graph growing algorithms, techniques employing neural networks [Rama Mohan Rao, Appa Rao and Dattaguru (1998) ; Pian , De Oliverira, Goddard (1999)], simulated annealing [Williams (1991); Bouhmala and Pahud (1998)], graph bisection algorithms like spectral bisection algorithms [Simon (1991); Barnard and Simon (1994)]. There is another class of algorithms, which are particularly popular, and successful in addressing this mesh partitioning problem, known as multilevel algorithms [Hendrickson and Leland, (1995); Karypis and Kumar (1999); Walshaw and Cross-, (1999)]. They usually combine a graph contraction algorithm with a local refinement algorithm and a heuristic fast coarse graph partitioning algorithm. Among all, spectral bisection and the multilevel algorithms have established a reputation for producing high-quality partitions. Apart from generating quality partitions, the multilevel algorithms are found to be extremely fast and they are primarily designed for ordering sparse matrices or for partitioning these matrices. As these sparse solvers are expected to take only few seconds, multilevel algorithms are effective to partition the sparse matrices without too much of additional overheads. A detailed review on these mesh partitioning techniques is given elsewhere [Rama Mohan Rao (2001)].

In contrast to sparse solvers, the finite element solutions for large models typically say for nonlinear or nonlinear dynamic analysis (where incremental and/or iterative form of solutions are usually employed) run time on parallel machines will be high. For example, the crash worthiness studies will generally be carried out for about 100,000 time steps [Plaskacz, Ramirez and Gupta (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 the finite element solution as these applications (nonlinear/nonlinear dynamics) require interprocessor communication in each time step/ iteration or increment. It can be also applied to the meshless methods [Atluri and Shen (2002), Atluri, Han and Shen (2003)].

The quality of generated partitions for an application can be assessed only after taking into consideration the architectural features of the parallel platform. These architectural features vary from machine to machine. The dedicated parallel processing machines like Intel PARAGON, IBM SP-2, Indian PARAM-10000 etc., are built with high-speed interconnection network (usually a proprietary switch). The communication overheads in these dedicated machines are less when compared to the cluster of workstations where the interprocessor communications are usually accomplished using the local area network. In such situations, the minimisation of interface nodes has greater thrust at the expense of slight imbalance in the computational load. Most of the heuristic algorithms reported in the literature including the multilevel approaches generate the partitions with minimum cut edges while maintaining the computational load approximately equal. None of these algorithms can cater to these mutually conflicting interests without modifications to the basic algorithm.

On the other hand the optimization based algorithms like gradient descent, simulated annealing, genetic algorithms (GAs) which are based on a cost function can be more effective in these circumstances. These optimization algorithms consider the cost function as a black box and the performance of the basic optimization algorithm will not be affected with changes in the cost function. The partitioning algorithms can easily be tuned to the individual requirements by simply modifying the cost function and they approach the problem more holistically than most of the heuristic algorithms reported in the literature. Apart from that, soft computing techniques like genetic algorithms have been proved to be successful for solving combinatorial optimisation problems [Park and Carter (1995)].

In this paper, an attempt has been made to devise a mesh partitioning technique employing genetic algorithms. The proposed algorithm has been devised employing float- encoded genetic algorithms. Unstructured meshes are considered as numerical examples and solved to demonstrate the effectiveness of the proposed GA based mesh partitioning algorithms.

2 Genetic Algorithms

Genetic algorithms (GAs) are general-purpose search algorithms, which use principles inspired by natural genetic populations to evolve solutions to problems [Holland (1975); Goldberg (1989)]. The basic idea is to maintain a population of chromosomes, which represent candidate solutions to the concrete problem that evolves over time through a process of competition and controlled variation. Each chromosome in the population has an associated fitness to determine which chromosomes are used to form new ones in the competition process, which is called selection. The new ones are created using genetic operators such as crossover and mutation. GAs have had a great measure of success in search and optimisation problems and have been employed for many combinatorial optimisation problems [Park and Carter, (1995)]. The reason for a great part of their success can be attributed to their ability to exploit the information accumulated about an initially unknown search space in order to bias subsequent searches into useful subspaces, i.e. their adaptation. This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools are inappropriate, offering a valid approach to problems requiring efficient and effective search techniques.

GA initially starts with a population of randomly generated chromosomes (genome), and advances toward better chromosome by applying genetic operators, modelled on the lines of genetic process occurring in nature. The population undergoes evolution in a form of natural selection. During successive iterations, called generations, chromosomes in the population are rated for their adaptation as solutions, and on the basis of these evaluations, a new population of chromosomes is formed using a selection mechanism and specific genetic operators such as crossover and mutation. An evaluation or cost (fitness) function H, must be devised for each problem to be solved. Given a particular chromosome, a solution, the fitness function returns a single numerical fitness, which is supposed to be proportional to the utility or adaptation of the solution, which that chromosome represents.

Although there are many possible variants of the basic GA, the fundamental underlying mechanism operates on a population of chromosomes or individuals, which represents possible solutions to the problem, and consists of three operations namely, evaluation of individual fitness, formation of a gene pool (intermediate population) through selection mechanism and finally recombination through crossover and mutation operators.

Representation is a key issue in GA because GAs directly manipulates a coded representation of the problem and because the representation schema can severely limit the window by which a system observes its world [Koza (1992)]. Fixed length and binary coded strings for the representation solution have dominated GA research since there are theoretical results that show them to be most appropriate ones [Goldberg (1991)]. Moreover binary representation is amenable to simple implementation. However, GA's good properties do not stem from the use of bit strings [Antonisse (1989) ; Radcliffe (1992)]. In view of this, several non-binary representations have been attempted based on application requirement and one of the most important among them is the floating-point number representation.

It is natural to represent the genes directly as floating point numbers for optimisation problems of parameters with variables in continuous domains. These are called float- encoded genetic algorithms and also referred to as real coded genetic algorithms in the literature. This representation makes it possible to use large domains for the variables, which is difficult to achieve in binary representation where increasing the domain would mean sacrificing precision, assuming fixed length for the chromosomes. Another advantage of the Float Encoded Genetic Algorithms (FEGA) is their capacity to exploit the *graduality* of the functions with continuous variables, where the concept of *graduality* refers to the fact that slight changes in the variables correspond to slight changes in the function. In this line, a highlighting advantage of FEGA is the capacity for the local tuning of the solutions. In view of this, FEGA have been employed in the present formulations. Moreover, Since the search space in the present formulation is continuous, FEGA is ideally suited.

3 Review of previous work

There are a limited number of papers about topics related to development of mesh partitioning techniques employing genetic algorithms. Khan and Topping [Khan and Topping, (1998)] have used genetic algorithms for partitioning the 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 the number of elements to appear in final refined sub-meshes. Mansour 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 direct coding has also been applied by Gil , Ortega, Diaz and Monotoya (1998) to graph partitioning in the context of circuit partitioning. However, the authors reported that GA based approach was outperformed by an approach devised synthesizing simulated annealing with tabu search. Soper, Walshaw and Cross (2001) have devised a graph partitioning technique by combining a multilevel algorithm with an evolutionary search procedure, which is reported to have better quality interms of cut edges with higher runtime. All the formulations reported in the literature employ Binary GA. In contrast, the proposed algorithm uses a distinct approach, with FEGA and also Micro FEGA.

4 Objective Function

The aim of any good mesh partitioning algorithm as already discussed is minimisation of the interprocessor communication and also maintaining equal distribution of computational load among processors. Therefore, the objective function for mesh partitioning can be written as:

$$\mathrm{H} = \mathrm{H}_{calc.} + \mu\, \mathrm{H}_{comm.} \tag{1}$$

Where H_{calc} represents load balance 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.

5 Problem Formulation

One of the effective ways of formulating the problem of mesh partitioning (i.e. partitioning of the associated graph) employing genetic algorithms is by combining them with multilevel algorithm [Rama Mohan Rao , Appa Rao and Dattaguru (2001)]. When GA is synthesized with multilevel algorithms, the genetic algorithms need to apply in each coarsened level. Since GA is relatively much slower than their compatriot local refinement algorithms like KL (Kernighan-Lin), Mob or other local refinement heuristics, the over all computational time is likely to be large. Moreover, in this formulation, the chromosome size is equal to the number of vertices in the graph, which may lead to memory and also convergence problems, while solving very large problems even with multilevel approaches. Apart from this, the multilevel algorithms are sensitive to the graph coarsening methods. Keeping all these things in view, a novel formulation has been employed in the present work.

The current formulation requires the positional parameters of the vertices. Since the algorithms are to be applied for partitioning finite element meshes, the positional parameters i.e., the (x, y, z) values of each vertex of the corresponding dual graph can be obtained by averaging the nodal coordinates of the corresponding element. Two arbitrary points are chosen within the extents of the domain as illustrated in Fig. 1 and are allowed to float around till the plane passing through these points bisects the graph into two good and equal partitions. This can be accomplished by bringing the analogy of the static electric field around a charged particle (Coulomb's law) [Giannakoglou and Giotis, (2001)]. The two arbitrary points can be thought of as a pair of point-charges floating around a domain. Each point charge (A or B) creates a 3-D scalar field around the domain. The field value at any graph vertex (x, y, z) can be determined by considering its distance from point A(x_A, y_A, z_A) and B (x_B, y_B, z_B). The following mathematical expression for the scalar value F at any graph vertex (x, y, z) can be employed.

$$\mathrm{F(x,y,z)} = \frac{\mathrm{K}_A}{\mathrm{r}_A^2} - \frac{\mathrm{K}_B}{\mathrm{r}_B^2} \tag{2}$$

where r_A and r_B are the distances of a vertex from the floating arbitrary points and given by

$$r_A = \sqrt{(x_A - x)^2 + (y_A - y)^2 + (z_A - z)^2} \qquad (3)$$

$$r_B = \sqrt{(x_B - x)^2 + (y_B - y)^2 + (z_B - z)^2} \qquad (4)$$

The scalar field value can be used as separator for partitioning the graph. K_A and K_B are two constants.

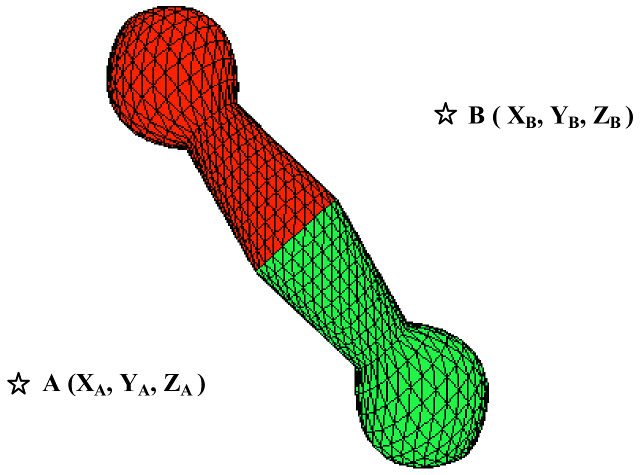

Figure 1 : Partitioning of a 3D domain using FEGA-GB

It is evident from the above discussion, that only eight parameters (x_A, y_A, z_A, x_B, y_B, z_B, k_A, k_B) are enough to obtain separators for partitioning the graph. The genetic algorithm controls these eight free parameters. The separators computed using these eight free parameters can be used to partition the graph into two equal parts. In other words for any pair of point charges, a single potential value is computed at any graph vertex. These potential values are sorted and the graph vertices above and below the median are assigned to the first and second submesh respectively. This satisfies the load-balancing requirement. In this case, the first term in the cost function given in Equation (1) becomes zero. Hence, only the second term of the objective function will be enough to compute fitness values. In order to reduce one more parameter, K_A is assumed as –1 and K_B is chosen arbitrarily.

6 Float-encoded genetic algorithm and genetic operators

In genetic algorithm, the decision variables are treated as organisms as a series of genes. These genes correspond to the chromosomes in natural genetics. The value of the objective function corresponding to the design vector provides a measure of the quality of fit of the vector. The aim of the GA is to find decision vector with the best fit (corresponding to the minimum value of the objective function).

In the present work, float-encoded genetic algorithms(FEGA) have been have employed to obtain the optimal (near optimal) parameters. Since proposed mesh partitioning algorithm deals with continuous variables and as the search space is continuous, FEGA is more effective than the standard GAs. The procedure for the float-encoded genetic algorithm employed in this paper, is described as follows:

6.1 Encoding

In the FEGA employed in this work, each parameter is represented as a floating-point decimal number with 15 digits. The float point representation of parameters in GAs has an another advantage as it avoids the difficult encoding and decoding of binary data that is required in standard GAs.

6.2 Selection

Selection embodies the principle of ‘Survival of the fittest’. The fitness of a particular design vector, termed individual, provides a measure of the suitability of the vector. A ‘good’ individual has high fitness while a ‘bad’ individual would have a low fitness. ‘Good’ individuals are selected for reproduction while ‘bad’ individuals are eliminated. For this selection process in GA, several selection operators have been proposed in the literature. Among them, roulette wheel and tournament selection operators are popularly being used in many applications. In the present work, a simple binary tournament selection has been employed as it is found to be effective, when compared to roulette wheel selection operator, while solving several test problems with the proposed formulations. The details of these studies have not been presented here as they are of least consequence.

In the binary tournament selection employed in the present work, pairs of individuals are picked at random from the population and the individual (genome) which has higher fitness is copied into the mating pool. This is repeated until the mating pool is full.

6.3 Crossover

An important feature of the genetic algorithms is known as crossover. Crossover consists of taking two selected chromosomes as parents and combining them with a certain probability to create two new children, which enter into a new population. Since the parameters are encoded as floating point numbers in FEGA, the conventional crossover and mutation operators being employed for binary (traditional) GA cannot be used.

In the present work, SBX crossover [Deb and Agarwal (1995)] has been employed. SBX crossover has been formulated by simulating the crossover operator of binary GA. In order to formulate SBX crossover, a probability distribution is used around parent solutions. For this purpose, the probability distribution is first calculated for single point crossover operator in binary GAs and the same is then adopted for FEGA. The probability distribution is represented as a function of a non-dimensional parameter β in order to be independent from parent solutions.

$$\beta = \frac{|c_2 - c_1|}{|p_2 - p_1|} \tag{5}$$

Where c_1 and c_2 are children solutions and p_1 and p_2 are parent solutions. The probability distribution is given as

$$\begin{aligned} P(\beta) &= 0.5(n+1)\beta^{n} \quad && \text{if } \beta \leq 1 \\ P(\beta) &= 0.5(n+1)\beta^{n+2} \quad && \text{if } \beta > 1 \end{aligned} \tag{6}$$

'n' is a parameter which controls the extent of spread in children solutions. Children solutions are created as near parent solutions when n is taken as large value. Smaller value of 'n' allows children solutions far away from parents. Moreover the probability distribution chosen for SBX preserves the following two observations found in crossover operators of binary GA:

1. The average of parent and children solutions is same i.e. it satisfies the condition $p_1 + p_2 = c_1(p_1, p_2) + c_2(p_1, p_2)$.
2. If the crossover is applied between two children individuals at the same cross site as was used to create children, the same parent individuals will result. This is preserved by assigning equal overall probability for creating solutions inside and outside the region enclosed by parent solutions.

The details of the implementation of crossover in the present work is as given in Fig. 2. The implementation ensures that children solutions are within the given domain (i.e. within the upper and lower limits of the variable).

$x_{mean} = (p_1 + p_2)/2$

$x_{diff} = p_1 - p_2$

$x_{dist} = p_1 - MIN$ or $MAX - p_2$ (whichever is smaller)

MIN and MAX are the lower and upper bounds of the particular variable under consideration

$\alpha = 1 + (2.0 * x_{dist} / x_{diff})$

$U_{max} = 1.0 - 0.5/\alpha^{(n+1)}$

'n' is parameter which controls the search space and is generally taken as 2

R = U_{max} * RAND(0, 1) where RAND is a random number generator for floating point numbers

if (R< =0.50) spread factor $\beta = (2.0 * R)^{1/(n+1)}$

if(R > 0.50) Spread factor $\beta = (0.5(1 - 2.0*R))^{1/(n+1)}$

$c_1(p_1, p_2) = x_{mean} + 0.50 * \beta * x_{diff}$

$c_2(p_1, p_2) = x_{mean} - 0.50 * \beta * x_{diff}$

Figure 2 : Crossover in float-encoded genetic algorithm

6.4 Mutation

Mutation allows new areas to be explored in the search space. If the mutation operation of binary GA is considered, the aim is to change the parameter 'c' of a certain individual to another individual 'c_1' in a given domain. The purpose of mutation operation for FEGA is same as for traditional binary GA. However, the traditional mutation operators being used in binary GA can not be employed here. Hence a parameter based mutation operator [Deb and Goyal (1996)] has been employed in the present work. A polynomial probability distribution is used to create a solution c_1 in the vicinity of a parent solution c. The details related to implementation of mutation operator are given in Fig. 3.

$d_1 = c - MIN$

$d_2 = MAX - c$

MIN and MAX are the lower and upper bounds of the particular variable under consideration

$$\delta = \frac{\min(d1,\ d2)}{(MAX - MIN)}$$

R = RAND (0, 1) where RAND is a random number generator for floating point numbers.

If $(R <= 0.50)\ \delta x = [2*R + (1-2*R)(1-\delta)^{1/K+1}]^{1/(k+1)} - 1$

If $(R > 0.50)\ \delta x = 1-[2*(1-R) + 2*(R-0.50)(1-\delta)^{K+1}]^{1/(k+1)}$

The value of K is set to $K \approx 100$, in order to a mutation effect of 1% perturbance in solutions

$\delta = \delta x * (MAX-MIN)$

$c_1 = c + \delta$ where c_1 is the new value of the variable after mutation.

Figure 3 : Mutation operator in float-encoded GA

7 Computational procedure

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 partitioning algorithm operates on the dual graph corresponding to the given finite element mesh. The genome consists of seven parameters as described above and the initial population is chosen randomly. The extents to the arbitrary points A and B are fixed based on the maximum and minimum values of the positional parameters of the graph vertices. The positional parameters of these two arbitrary points are chosen randomly within these fixed extents.

The field separators are computed using the Equation (2) and the vertices are partitioned based on this field separator. The number of cutedges N_{ce} can be evaluated by employing the Laplacian matrix, L of the graph

$$N_{ce} = x^T L x \qquad (7)$$

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

The complete procedure for FEGA is described as follows:

1. Encode each parameter to float decimal number.
2. Specify certain population size and maximum number of generations, and the number of runs (MR), then generate an initial population randomly.
3. Evaluate the fitness value of each individual (using equation (2) for field separators and equation (7) for cutedges).
4. If convergence condition is satisfied, then print the optimal solutions else continue.
5. Perform the selection operation (binary tournament selection).
6. Perform the crossover and mutation operations.
7. Repeat steps (3) – (7) until the maximum number of generations is completed or convergence is achieved.
8. Repeat steps (1) – (7) for each run till maximum number of runs is completed.

This method is termed as FEGA-GB (Float-encoded Genetic Algorithm for Graph Bisection). It is appropriate to point out here that the computational cost of this method is proportional to the graph size (i.e. number of vertices in the graph). For large size graphs, the run time is expected to be quite high. In view of this, attempts have been made to reduce the partitioning cost by devising certain acceleration-refinement techniques. The various acceleration-refinement techniques employed are discussed here.

7.1 Method FEGA-GB(S)

In this formulation, bisection is based on a potential field created by point charges. The potential field is always continuous and defines more or less distinct subgraphs, even during the very first generation. Hence, the number of generations required for convergence is usually low. However, the computations involved in each generation are high making the total computational time very high for large graphs. In order to accelerate the convergence process, the following approach has been employed.

1. Run GA (FEGA-GB) only for two/three generations
2. Compute the average fitness of the population.
3. Shrink the search space of all free parameters based on the values corresponding to the population with best fitness. The search space is centered upon the location of the fittest solution and extends symmetrically in each direction by a user-defined factor. The shrinking of the search space is carried out at a user-defined interval.
4. Repeat steps (1) to (4) till convergence

User-defined factor indicates to what extent, the search space should extend symmetrically in each direction from the location of the fittest solution (i.e. from optimal points A and B shown in Fig.1). In other words, the user-defined factor can be defined as the ratio of 'minimum distance between the optimal point (A or B in Fig.1) and the extended point' to 'the minimum distance between the optimal point and upper/lower limit point'. This factor has been taken as 0.25 in the present work.

The shrinking of search space can as well be carried out in each generation. However little purpose will be served in doing so. Moreover, it involves some extraneous calculations in fixing the revised search space. In view of this it is proposed to shrink the search space only after every specific number of generations defined by a parameter called 'user defined interval'. In the present work this has been set to 4.

7.2 Method FEGA-GB (I)

The compute intensive part of the formulation being discussed in this paper is the computation of matrix vector product $x^T Lx$ and subsequent sorting. Eventhough, the Laplacian, L is not formed explicitly and the sparsity of the matrix is taken into account while performing matrix vector products, this operation is still a compute intensive one. These computations are proportional to the number of vertices in the graph. For larger graphs, this partitioning process will obviously be very slow and undesirable. Any economy achieved in computing the matrix vector product, result in improvement in the performance of the algorithm. In order to achieve this, the following procedure is employed.

1. Run GA (FEGA-GB) for two or three generations considering the whole graph to be partitioned
2. Select the bisection corresponding to the fittest individual of the population. These separators are expected to be reasonably good and the separator can be further improved by migrating interface vertices across the boundary i.e., local refinement.
3. Mark the interface vertices. The idea here is to choose only the vertices, which are most likely to swap across partitions. These are termed as '*active vertices'* hereafter. The rest of the vertices are expected to lie in the same partition during the subsequent generations and hence will be called as '*inactive vertices'*. Once these vertices are identified, the computations can be drastically reduced.
4. Compute the matrix vector product $x^T Lx$ corresponding to the inactive nodes and store as Cu_i.
5. Apply GA only on the active vertices. Each time, the matrix vector product $x^T Lx$ corresponding to the active vertices is computed and added to the Cu_i to get the total cutedges of the bisection. The sorting is also confined only to the active vertices. Here, both sorting as well as matrix–vector products are just confined only to the active vertices. These active vertices will generally be a small fraction of the total graph. This is likely to save computing time especially while solving large size problems.

7.3 Method FEGA-GB (I-S)

This method is a combination of FEGA-GB (I) and FEGA-GB (S). It can be described as follows:

1. Run GA (FEGA-GB) for two or three generations by considering the whole graph to be bisected.
2. Select the bisection corresponding to the fittest individual of the population. Choose the active vertices corresponding to the interfaces.
3. Apply GA on these active vertices for a selected number of generations say four or five generations.
4. Define a new search space based on the values of free parameters of the current fittest solution. The search space is centered upon the location of the fittest solution and extends symmetrically in each direction by a user-defined factor. The adaptive shrinking of search space is carried out at every user-defined interval.

5. Apply GA on the active vertices and considering only the shrunken search space.

7.4 Micro float-encoded Genetic Algorithms (μ-FEGA) For Graph Partitioning

The genetic algorithms are certainly useful for obtaining robust solutions for mesh partitioning problems. However, the time penalties involved in evaluating the fitness functions are quite high especially for large size problems. In order to reduce the computational time, the micro-genetic algorithms have been employed.

The term micro-genetic algorithm (micro-GA) refers to a small population genetic algorithm with reinitialization. The idea was first suggested by Goldberg (1989) based on some theoretical results. According to him, just a population size of 3 was sufficient to converge, irrespective of the size of the chromosome. The process suggested by Goldberg was to start with a small randomly generated population, then apply to it the genetic operators until reaching *nominal convergence* and then to generate a new population by transferring the best individuals of the converged population to the new one. The remaining individuals would be randomly generated. The application based on micro-GA was reported later by Krishna Kumar (1989). He has applied for the problem of non-stationary function optimization. He has used population size as 5, a crossover rate of 1 and mutation rate of zero. The results obtained was reported to be superior to standard GA. The same concepts of micro-GA have been applied here for graph partitioning, with some modifications and are termed as micro float-encoded genetic algorithms (μ-FEGA). The μ-FEGA implementation is as discussed below:

1. Perform genetic operations on the total population for two or three generations. Total population is considered as 40 in our experiments.
2. Select the new population size as six. Choose two copies of best ever string till the previous generation and then randomly pick four other from the shuffled total population (i.e., forty which were generated in step (1)).
3. Evaluate the fitness and determine the best string. Carry two copies of the best string to the next generation (elitist strategy). This ensures that the information about good schema is preserved.
4. Choose the remaining four strings for reproduction based on a deterministic tournament selection. The best string also competes for a copy in the reproduction. Apply crossover, mutation and evaluate the fitness function.
5. Check for convergence.
6. If not converged Go to step 2.
7. If converged, preserve the best solution to use in the next run. Repeat steps (1) to (5) for the next run. The maximum runs are taken as four in our numerical experiments.

Since the objective here is to find the best ever solution as quickly as possible rather than the best average behavior, micro-float-encoded genetic algorithms (μ-FEGA) are expected to work efficiently.

In order to further accelerate the computational process, the μ-FEGA is employed both with shrinking the search space (μ-FEGA-(S)), with active vertices (μ-FEGA-(I)) and with both shrinking and active vertices(μ-FEGA-(S-I)). Numerical studies to demonstrate various formulations discussed above are presented in the next section.

8 Evaluation of mesh partitioning algorithms

In this paper, mostly unstructured meshes are considered as numerical examples mainly due to two reasons. First, the unstructured meshes are often better suited than regular structured grids for representing completely general geometries and resolving wide variations in behavior via variable mesh densities and are becoming quite popular among the users of the finite element and finite volume methods. The second reason is that partitioning of these unstructured meshes is more cumbersome and challenging.

The GA based mesh partitioning algorithms have been integrated into PSTRAIN (Parallel STRuctural Analysis INterface) [Rama Mohan Rao (2001)] software, which consists of variety of mesh partitioning algorithms. PSTRAIN is built with powerful Graphic User Interface using X, Motif and OpenGL in order to facilitate visualisation. The proposed algorithms have been implemented on Unix workstations. For the numerical studies reported in this paper HP C-240 workstation is used.

The eight variants of mesh partitioning algorithms developed using Float-encoded GA have been employed to solve the unstructured meshes given in Fig. 4 and Fig. 5. The population size is taken as 40 for FEGA and 6 for μ-FEGA. SBX crossover with a probability of 80% has been employed for FEGA and 90% for the μ-FEGA. The mutation is considered as 0.25% for the FEGA and it is set to 0.20% for μ-FEGA. The number of runs is one for FEGA and 4 for μ-FEGA.

Convergence studies have been carried out employing the proposed GA based mesh partitioning algorithms in order to optimally choose the convergence criterion. For this purpose, the two unstructured meshes have been solved employing different convergence criterion employing both FEGA and μ-FEGA. Since the convergence trend in both FEGA and μ-FEGA found to be same, only the results for μ-FEGA are presented here.

Generations for Convergence	4	20	40	50	80
Num. of Runs	4	4	4	4	4
Cut Edges	155	155	150	146	146
Total Num. of Generations	38	187	352	474	508

Table 1 : Convergence of μ-FEGA for unstructured mesh of JOINT1 for generating four partitions

Generations for Convergence	4	20	40	50	80
Num. of Runs	4	4	4	4	4
Cut Edges	306	302	293	289	289
Total Num. of Generations	86	435	695	1046	1121

Table 2 : Convergence of μ-FEGA for unstructured mesh of JOINT1 for generating eight partitions

Tab. 1 to 4 show the results of μ-FEGA, while generating four and eight partitions for the two unstructured meshes shown in Fig. 4 and 5. The numbers indicated against the field '*convergence*' in the tables indicate the number of consecutive generations considered as convergence criteria i.e., the FEGA is assumed to have been converged if there is no improvement in the solution in specified number of consecutive generations.

Generations for Convergence	4	20	40	50	80
Num. of Runs	4	4	4	4	4
Cut Edges	186	178	178	178	178
Total Num. of Generations	39	198	378	468	527

Table 3 : Convergence of μ- FEGA for unstructured mesh of JOINT2 for generating four partitions

Generations for Convergence	4	20	40	50	80
Num. of Runs	4	4	4	4	4
Cut Edges	339	328	323	323	323
Total Num. of Generations	90	499	859	987	1077

Table 4 : Convergence of μ-FEGA for unstructured mesh of JOINT2 for generating eight partitions

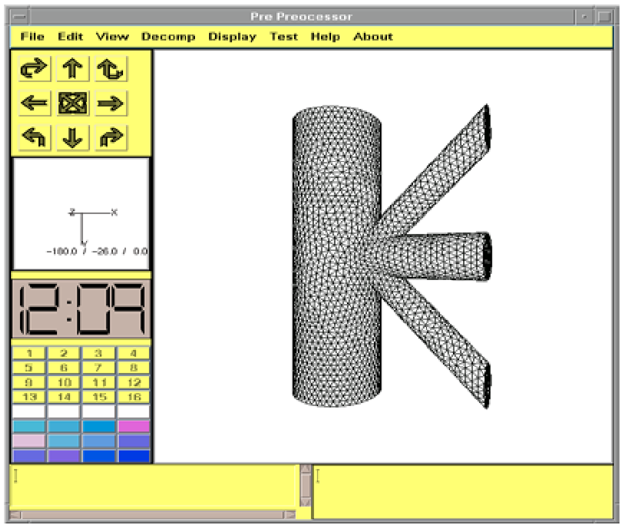

Figure 4 : Unstructured mesh describing a typical joint of an off-shore structure-JOINT1

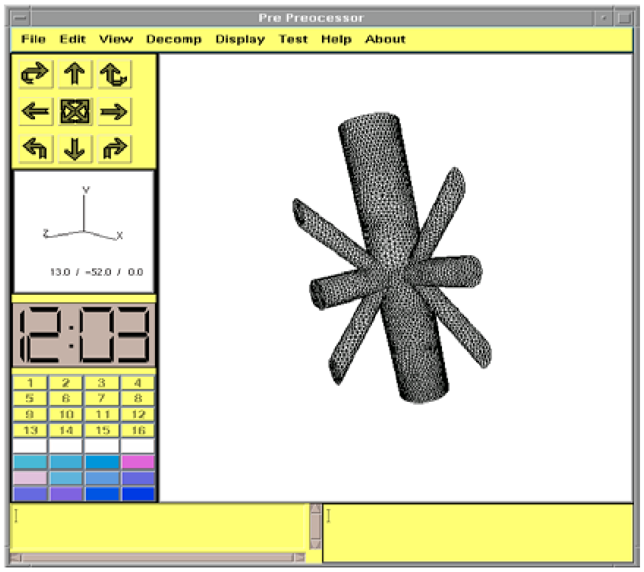

Figure 5 : Unstructured mesh describing a complex joint in a space frame - JOINT2

A close look at the results given in Tab. 1 to 4 indicate that there is a marginal decrease in cutedges, when the number of consecutive generations for convergence requirement is increased. At the same time, there is considerable increase in terms of total number of generations required for meeting the stringent convergence criteria. This obviously results in higher computational time. Similar trends have been observed with the proposed formulations for many other test graphs. In view of this, a rather relaxed convergence of four generations has been adopted through out the studies reported in this paper. i.e., the solution is assumed as converged if no improvement in solution is found in the last four consecutive generations.

The two unstructured meshes shown in Fig. 4 and 5 are solved for generating varied number of submeshes, employing various alternative FEGA implementations. The result interms of execution time (CPU in seconds) and cut edges (EC) for the two examples are shown in Tab. 5 to 8.

In order to solve a moderately large problem, an unstructured mesh of an automobile body (BODY) is taken as a numerical example. The number of vertices and edges in the associated dual graph of the unstructured mesh are 45087 and 163734 respectively. The results obtained from all the variants of FEGA and μ-FEGA are presented in Tab. 9 and 10 respectively.

It is evident from the results, that the FEGA-GB generates partitions with minimum cut edges. It is however takes longer time to converge. The interface refinement occasionally improved the results with lesser computational cost. For smaller problems the saving in computational cost appears to be marginal. However, for even moderate size problems (Tab. 9), interface refinement can save up to 14% of computing time. The shrinking of extents has an effect on the acceleration of the solution. However, the savings in the cost are marginal. This may be due to very relaxed convergence criteria employed in order to accelerate the runtime of the algorithm. The results obtained by μ-FEGA are slightly inferior to the FEGA-GB. However, the cost of computation has been drastically reduced. It is worth mentioning here that no special efforts have been made to tune the parameters for micro-float-encoded genetic algorithms (μ -FEGA) for optimal performance. Hence, with optimal tuning of the parameters, the results obtained using μ -FEGA are likely to improve further.

Finally, the proposed GA based algorithms have been evaluated by solving several benchmark graphs of finite element meshes available in the net. Since the proposed algorithms require coordinate information, only the benchmark graphs available with positional parameters are considered for evaluation. The details of these graphs are given in Tab. 11. Here only μ-FEGA based partitioning algorithms, which are fastest among the GA based algorithms discussed in this paper, are considered for comparison. These algorithms have been compared with the multilevel spectral bisection algorithm, RSBM [Bernard and Simon, (1994)] and multilevel graph partitioning algorithm, METIS [Karypis and Kumar (1998,1999)].

For this purpose, a multilevel spectral algorithm with boundary KL (Kernighan-Lin) refinement and METIS version 4.0 available in the net are implemented on HP C-240 workstation. METIS provides two algorithms PMETIS and KMETIS for partitioning an irregular graph into k equal size parts. PMETIS [Karypis and Kumar, (1999)] uses the multilevel paradigm and solves the k-way partitioning of the graph by recursive bisection. In contrast, KMETIS [Karypis and Kumar, (1998)] uses multilevel paradigm to construct a k-way partitioning of the graph directly i.e. the coarsest graph is directly partitioned into k parts and this k-partitioning is refined suc-

NP	FEGA-GB		FEGA-GB(S)		FEGA-GB(I)		FEGA-GB(I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	6.1	66	5.15	66	8.3	66	7.6	66
4	11.6	164	10.2	164	8.9	164	8.2	164
8	16.7	294	14.8	294	13.0	291	12.2	291
16	21.7	448	19.2	448	18.9	432	16.6	427
32	26.5	624	22.8	624	23.7	631	20.9	631
64	31.2	863	28.7	863	28.7	851	25.8	851

Table 5 : Performance of FEGA-GB based partitioning algorithms- JOINT1

NP	$\mu-$FEGA		$\mu-$FEGA (S)		$\mu-$FEGA (I)		$\mu-$FEGA (I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	1.64	68	1.23	68	1.25	68	1.16	68
4	3.10	155	2.28	155	2.36	155	2.11	153
8	4.57	304	4.22	310	4.56	304	5.16	304
16	5.9	485	5.1	485	5.02	485	4.86	485
32	7.22	713	6.02	727	6.24	713	6.08	713
64	8.57	965	7.16	968	7.59	968	7.07	968

Table 6 : Performance of μ-FEGA based partitioning algorithms- JOINT1

NP	FEGA-GB		FEGA-GB(S)		FEGA-GB(I)		FEGA-GB(I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	10.5	86	8.9	86	9.2	86	8.7	86
4	20.0	180	18.3	180	19.1	178	18.4	178
8	28.8	349	26.7	349	26.6	354	26.4	354
16	37.3	560	34.5	560	32.3	595	31.8	595
32	45.4	780	43.2	780	44.1	825	42.1	825
64	53.3	1070	49.8	1070	50.9	1073	48.8	1073

Table 7 : Performance of FEGA-GB based partitioning algorithms- JOINT2

NP	$\mu-$FEGA		$\mu-$FEGA (S)		$\mu-$FEGA (I)		$\mu-$FEGA (I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	2.84	90	2.44	90	2.76	88	2.54	88
4	4.52	186	4.02	186	4.20	184	4.08	184
8	7.88	339	7.21	340	7.60	344	7.41	344
16	10.2	575	9.98	575	9.92	582	9.54	582
32	12.5	879	11.0	882	10.0	879	9.82	879
64	14.6	1300	13.1	1302	14.2	1314	14.0	1314

Table 8 : Performance of μ-FEGA based partitioning algorithms- JOINT2

NP	FEGA-GB		FEGA-GB(S)		FEGA-GB(I)		FEGA-GB(I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	77	266	78	274	65	260	68	260
4	138	659	134	675	122	659	118	668
8	202	1184	190	1202	177	1180	173	1184
16	242	1914	235	1914	214	1914	210	1914
32	282	3422	287	3443	250	3412	248	3414
64	314	5018	309	5012	279	5004	281	5010

Table 9 : Performance of FEGA-GB based partitioning algorithms- BODY

NP	$\mu-$FEGA		$\mu-$FEGA (S)		$\mu-$FEGA (I)		$\mu-$FEGA (I-S)	
	CPU	EC	CPU	EC	CPU	EC	CPU	EC
2	8.9	272	9.2	272	8.2	270	8.5	270
4	16.1	662	15.2	662	12.2	658	12.6	662
8	23.8	1202	24.4	1239	18.6	1196	17.8	1196
16	31.2	1945	29.8	1945	26.4	1912	25.2	1926
32	35.6	3420	32.4	3418	31.2	3442	32.0	3418
64	43.4	5031	44.0	5018	38.6	5018	37.4	5018

Table 10 : Performance of μ-FEGA based partitioning algorithms- BODY

S.NO	Graph name	Num. of Vertices	Num. of Edges	Description
1	3ELT	9000	13278	2D Finite element mesh
2	AEROFOIL	8034	11813	2D CFD mesh
3	BIG	30269	44949	2D Finite element mesh
4	CRACK	20141	30043	2D Finite Element mesh
5	WHITAKER	19190	28581	2D finite Element mesh
6	FE-4ELT	15606	45878	2D Finite Element Mesh
7	BRACK2	62631	366559	3D Finite Element Mesh
8	FEMESH	19334	55305	3D Finite Element Mesh
9	BIPLANE	21701	42038	NASA bench mark problem
10	SHOCK	36476	71290	NASA bench mark problem

Table 11 : Finite element meshes for evaluation of FEGA based algorithms

NP		2	4	8	16	32	64
3ELT	$\mu-$FEGA	59	109	179	312	503	793
	NGEN	12	38	86	185	379	945
	Time	1.96	3.85	5.40	7.2	9.0	10.56
	RSBM	76	249	430	689	1173	1828
	Time	0.39	0.59	0.71	0.77	0.78	0.82
	PMETIS	65	156	243	358	569	907
	Time	0.19	0.21	0.24	0.28	0.33	0.38
	KMETIS	64	128	195	328	566	885
	Time	0.15	0.16	0.18	0.19	0.24	0.32
AIRFOIL1	$\mu-$FEGA	42	87	143	240	428	720
	NGEN	15	38	90	186	380	763
	Time	1.66	3.16	4.46	5.80	7.38	8.67
	RSBM	56	201	357	594	1042	1672
	Time	0.32	0.50	0.58	0.63	0.64	0.66
	PMETIS	39	88	156	285	487	820
	Time	0.15	0.16	0.19	0.21	0.25	0.29
	KMETIS	39	103	157	276	505	809
	Time	0.15	0.16	0.17	0.19	0.21	0.23
BIG	$\mu-$FEGA	138	242	548	1020	1904	2203
	NGEN	12	38	90	198	408	809
	Time	4.42	8.46	12.26	15.88	19.46	22.78
	RSBM	422	704	1231	1921	2489	3083
	Time	4.17	6.26	7.32	7.87	8.16	8.32
	PMETIS	87	209	358	585	962	1522
	Time	0.57	0.69	0.78	0.87	0.98	1.10
	KMETIS	77	189	334	603	901	1507
	Time	0.55	0.56	0.57	0.58	0.62	0.67
CRACK	$\mu-$FEGA	112	246	404	721	1014	1368
	NGEN	13	33	87	185	381	771
	Time	3.87	7.44	10.78	13.95	17.00	20.04
	RSBM	148	491	822	1302	1965	2964
	Time	1.8	2.7	3.17	3.40	3.54	3.62
	PMETIS	101	216	363	634	916	1382
	Time	0.37	0.45	0.51	0.56	0.63	0.71
	KMETIS	100	224	377	575	891	1313
	Time	0.36	0.37	0.38	0.40	0.42	0.48
WHITAKERR 3	$\mu-$FEGA	64	156	366	624	911	1352
	NGEN	12	37	89	189	385	765
	Time	3.57	6.90	10.05	12.9	15.9	18.78
	RSBM	88	398	704	1224	1825	2800
	Time	1.66	2.48	2.91	3.12	3.24	3.32
	PMETIS	70	207	368	650	974	1500
	Time	0.35	0.42	0.48	0.53	0.60	0.68
	KMETIS	73	213	382	643	970	1472
	Time	0.35	0.34	0.35	0.37	0.40	0.45

Table12: Performance (cut edges, EC) of μ-FEGA based mesh partitioning algorithms (NGEN: number of generations; Time: CPU time in seconds)

NP		2	4	8	16	32	64
SHOCK	$\mu-$FEGA	128	352	687	1131	1755	2629
	NGEN	12	38	86	187	374	758
	Time	8.04	15.5	23.0	30.9	38.7	46.7
	RSBM	156	448	956	1519	2270	3365
	Time	1.80	2.70	2.85	3.40	3.54	3.62
	PMETIS	151	399	761	1184	1946	2892
	Time	0.84	1.01	1.17	1.31	1.43	1.57
	KMETIS	147	420	706	1217	1930	2896
	Time	0.80	0.81	0.82	0.85	0.89	0.99
BIPLANE	$\mu-$FEGA	130	219	449	724	1169	1792
	NGEN	9	35	88	186	380	766
	Time	4.53	8.85	14.31	19.08	22.77	27.24
	RSBM	118	210	437	846	1413	2132
	Time	8.1	12.1	14.1	15.2	15.7	16.0
	PMETIS	82	189	440	812	1307	1998
	Time	0.47	0.56	0.64	0.71	0.78	0.88
	KMETIS	94	224	522	796	1260	1942
	Time	0.46	0.47	0.48	0.49	0.53	0.59
FE-4ELT	$\mu-$FEGA	177	347	672	1072	1704	2816
	NGEN	12	36	84	180	372	760
	Time	2.01	3.44	5.11	6.44	7.60	8.57
	RSBM	287	422	707	1232	1926	3087
	Time	1.89	2.71	3.17	3.40	3.54	3.62
	PMETIS	183	401	734	1101	1806	2905
	Time	0.45	0.51	0.57	0.62	0.69	0.78
	KMETIS	151	415	638	1092	1819	2869
	Time	0.43	0.43	0.44	0.46	0.48	0.54
BRACK2	$\mu-$FEGA	731	3008	7524	11092	20005	29354
	NGEN	12	42	92	197	417	837
	Time	14.6	28.8	40.9	52.6	63.8	78.3
	RSBM	1036	4125	8776	15404	22890	32999
	Time	5.10	6.89	10.28	11.95	12.80	13.78
	PMETIS	748	3230	7707	12838	20039	29232
	Time	3.41	4.05	4.59	5.08	5.56	6.01
	KMETIS	794	3251	8310	13068	19849	28969
	Time	3.31	3.34	3.36	3.42	3.51	3.69
FEMESH	$\mu-$FEGA	243	662	1708	2756	3549	4034
	NGEN	13	41	97	198	418	822
	Time	13.1	25.9	37.44	48.1	58.3	68.7
	RSBM	389	1292	2272	4570	6499	9627
	Time	2.44	3.22	5.98	7.01	8.42	9.67
	PMETIS	267	886	1970	3253	4688	6945
	Time	0.56	0.65	0.76	0.83	0.95	1.07
	KMETIS	300	950	2152	3599	5204	7180
	Time	0.54	0.55	0.57	0.60	0.64	0.74

Table 12: Performance (cut edges, EC) of μ-FEGA based mesh partitioning algorithms (continued)
(NGEN: number of generations; Time: CPU time in seconds)

cessively as the graph is uncoarsened back into the original graph.

The cutedges and CPU timings of μ-FEGA, RSBM, PMETIS, KMETIS while solving the benchmark graphs are presented in Tab. 12. A close look at the results indicate that in most of the situations, the cut edges generated employing μ-FEGA found to be much lesser than the two other popular mesh partitioning algorithms. However, the computational cost is quite high.

In order to demonstrate the effectiveness of the proposed algorithms through visuals, a few numerical examples are considered and results are compared with RSBM and METIS. First, the unstructured mesh of a typical joint (JOINT1) of an offshore platform given in Fig. 4 is solved employing μ-FEGA, PMETIS and RSBM and the partitioning results are shown in Figs. 6, 7 and 8 respectively. The associated dual graph has 7860 vertices and 11790 edges. It can be observed that the proposed GA based algorithm generates compact partitions without any domain splitting (i.e. the domains (submeshes) formed with discontinuous regions), while the other two algorithms generates fairly elongated partitions. Domain splitting can be observed in the partitions generated with METIS. The cutedges obtained for the four submeshes generated by μ-FEGA, PMETIS and RSBM are 155, 184, 191 respectively.

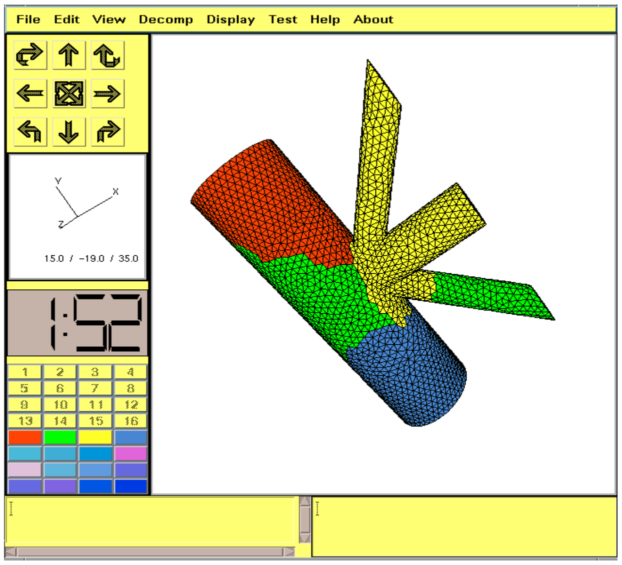

Figure 7 : Generation of four submeshes of JOINT1 employing PMETIS.

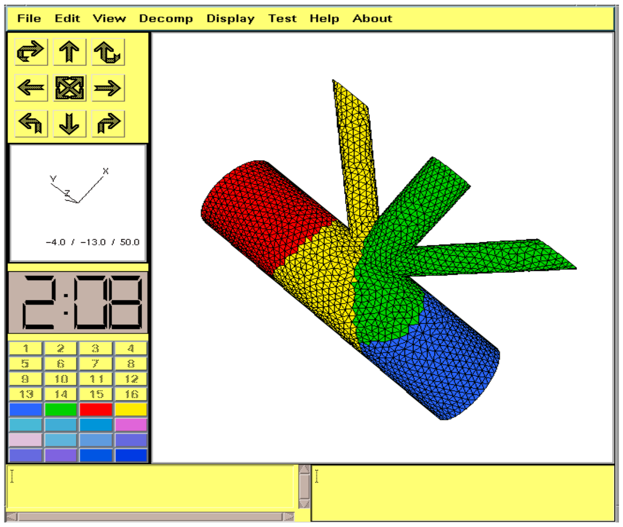

Figure 6 : Generation of four submeshes of JOINT1 employing μ-FEGA based algorithm

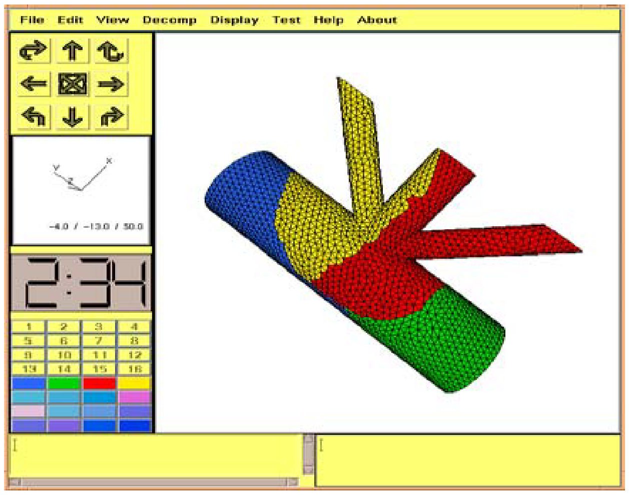

Figure 8 : Generation of four submeshes of JOINT1 employing RSBM.

Similarly, the partitioning results obtained for the unstructured mesh of a typical joint (JOINT2) shown in Fig.

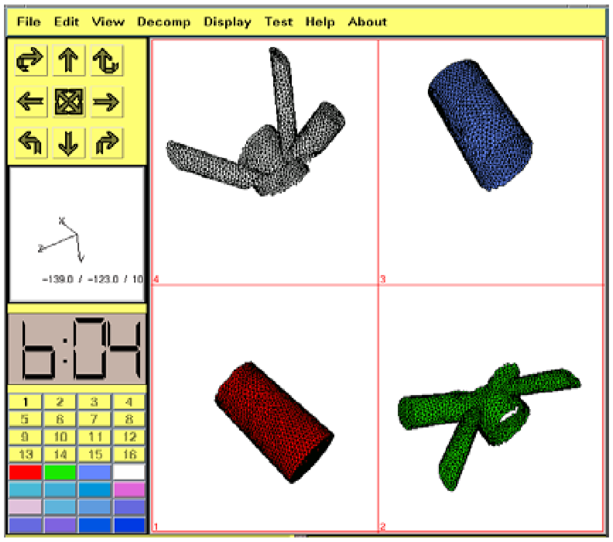

Figure 9 : Generation of four submeshes of JOINT2 employing μ-FEGA based algorithm

5 of a space frame are presented in Figs. 9-11. The associated dual graph consists of 12900 vertices and 19350 edges. A close look at the figures indicate that the proposed GA based algorithm exhibits better performance interms of partitioning the mesh. Split domains can be observed in the partitions generated by RSBM (Fig.11). The cutedges obtained for generating four submeshes by μ-FEGA, PMETIS and RSBM are 180, 186, 198 respectively.

An unstructured mesh describing a mechanical shaft, shown in Fig. 12 is considered as third numerical example. The associated dual graph consists of 4060 vertices and 5992 edges. Four submeshes are generated employing μ-FEGA, PMETIS and RSBM algorithms and the partitioning information is shown in Figs. 13-15. Here also one can observe the superiority of μ-FEGA in terms of generating partitions with optimal cuts. Domain splitting can be observed in the partitions generated by PMETIS (Fig. 14). The cut edges for four partitions are found to be 62, 69 and 55 for RSBM, PMETIS and μ-FEGA respectively.

Fig. 16 shows a 3D well graded finite element mesh (FEMESH) of a thick metal sheet. Four partitions generated employing μ-FEGA based partitioning algorithm are shown in Fig. 17. It can be observed that the partitions generated by GA based algorithms are quite compact with good aspect ratio. No domain splitting is ob-

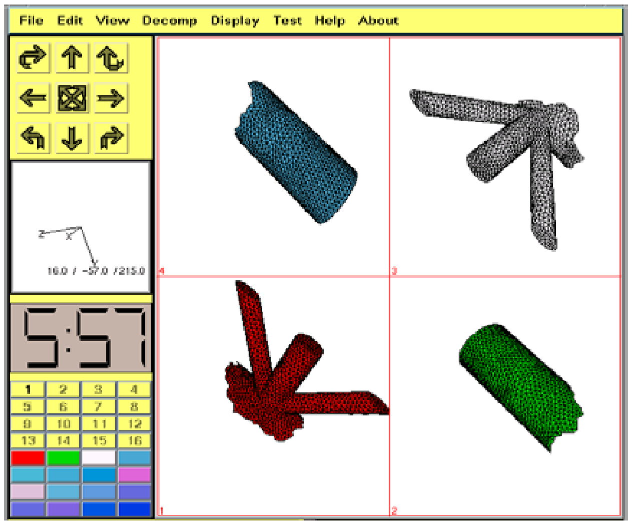

Figure 10 : Generation of four submeshes of JOINT2 employing PMETIS

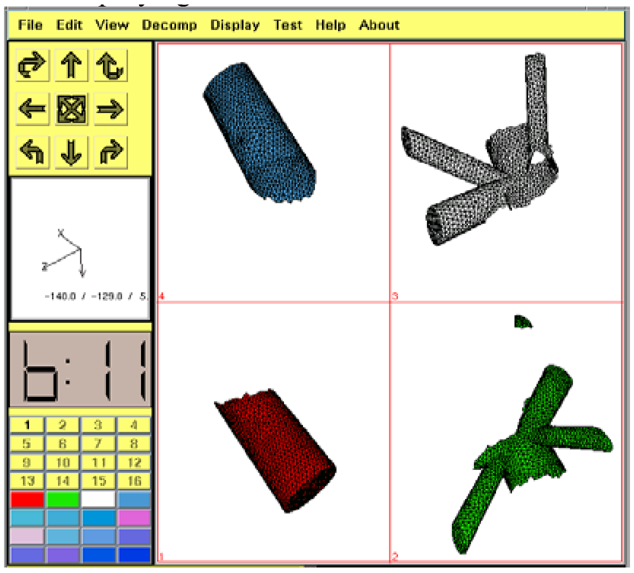

Figure 11 : Generation of four submeshes of JOINT2 employing RSBM

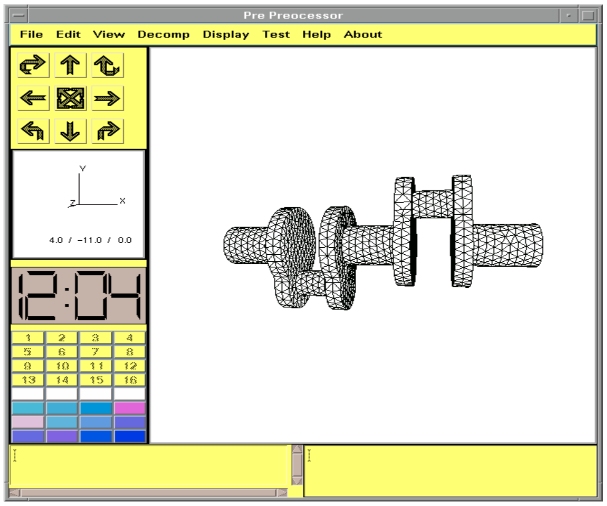

Figure 12 : Unstructured mesh describing a mechanical shaft (SHAFT)

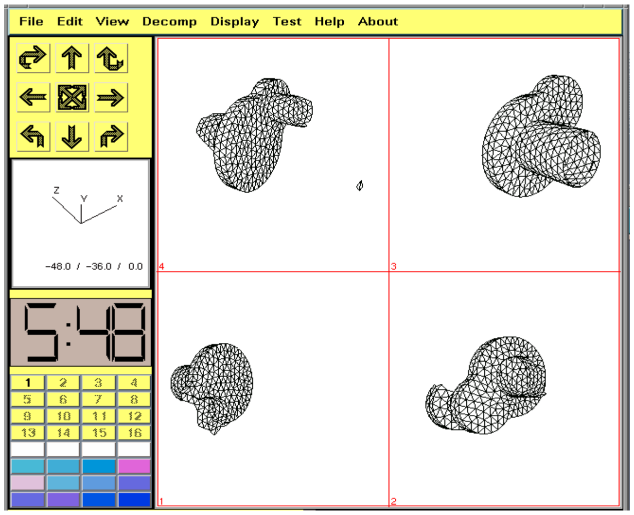

Figure 14 : Generation of four submeshes of SHAFT employing PMETIS

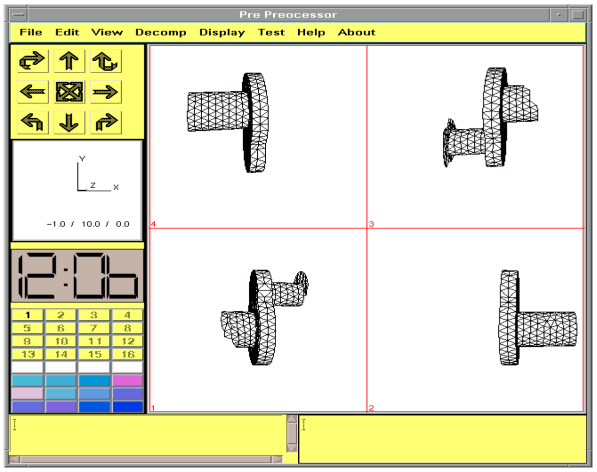

Figure 13 : Generation of four submeshes of SHAFT employing μ-FEGA based algorithms

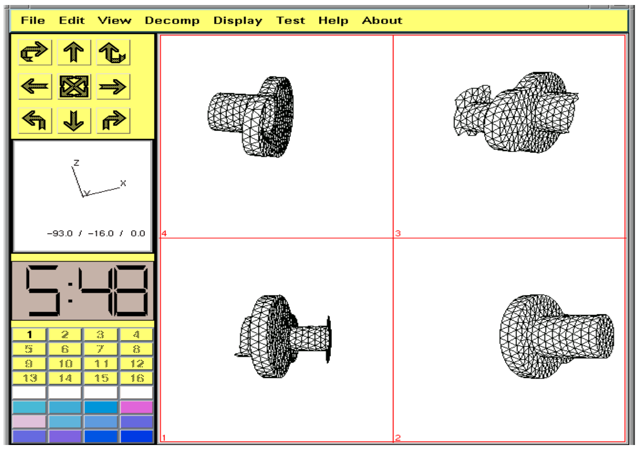

Figure 15 : Generation of four submeshes of SHAFT employing RSBM

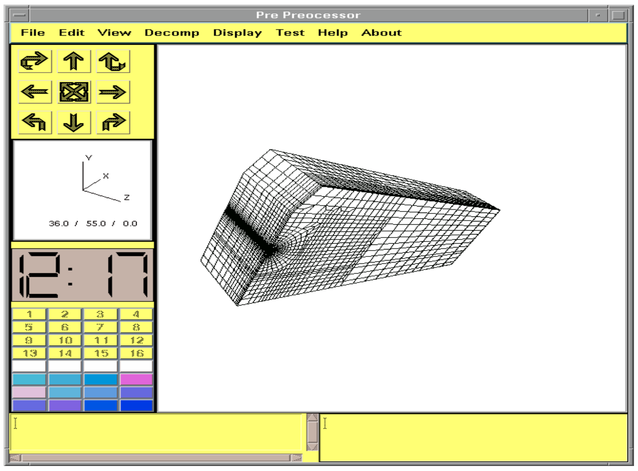

Figure 16 : A 3D graded mesh describing a thick metal sheet (FEMESH)

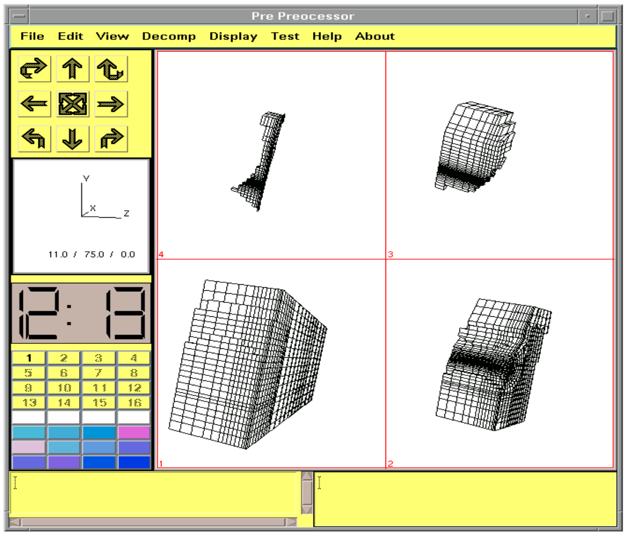

Figure 17 : Generation of four submeshes of SHAFT employing μ-FEGA based partitioning algorithm

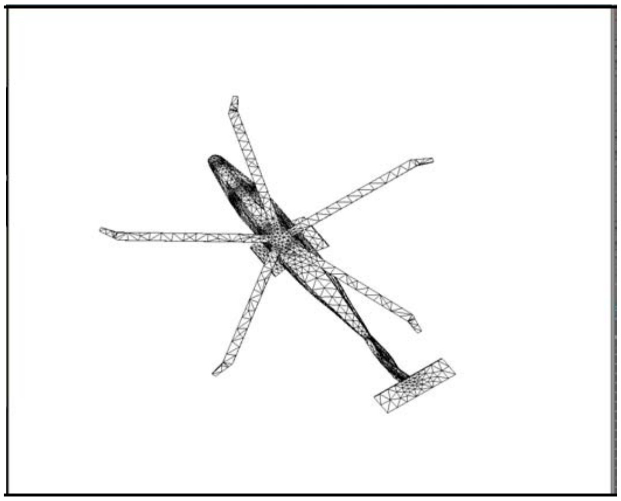

Figure 18 : Unstructured mesh describing the body of an helicopter(HELIC)

served in this case also.

Finally Fig. 18 shows an unstructured finite element mesh describing the body of an helicopter. The associated dual graph consists of 7920 vertices and 11654 edges. Fig. 19 presents the compact submeshes generated employing μ-FEGA based algorithm. The number of cut edges for eight partitions generated is found to be 321 against 654 of RSBM.

It can be observed from the results that the μ-FEGA based partitioning algorithms consistently generate better partitions than the multilevel algorithms like RSBM and METIS. The computational cost of μ-FEGA is higher when compared to the multilevel algorithms. However, GA based algorithms are excellent candidates for parallel processing than the multilevel or spectral approaches. The computational efficiency of these GA based algorithms can be substantially improved by moving these algorithms onto parallel processing machines.

9 Conclusions

A novel formulation has been proposed for addressing the mesh partitioning problem. The mesh partitioning problem has been formulated by allowing two arbitrary points to float within the graph associated with the given finite element mesh and optimising the position of these two arbitrary points. The arbitrary points are at their optimal positions within the extents of the graph, if the

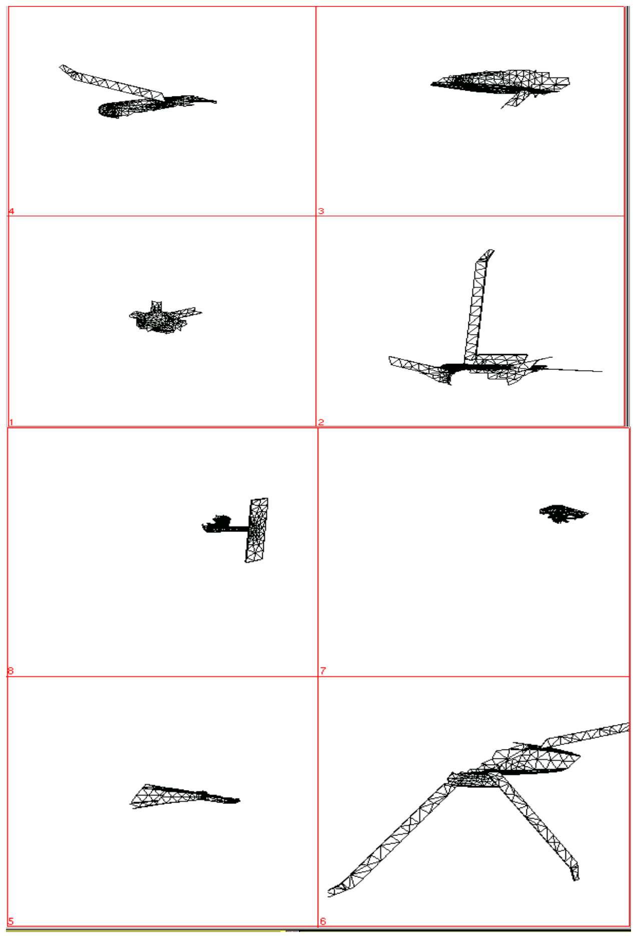

Figure 19 : Generation of eight submeshes of a mesh, HELIC employing μ-FEGA based algorithm

vertex separators computed with respect to the position of these arbitrary points, partition the targeted graph in such a way that the cutedges are minimum. Optimising the position of the two points is accomplished by employing genetic algorithms. Here the size of the genome is not dependent on the size of the graph and only seven parameters need to be considered while constructing the genome. Since the parameters are continuous, float-encoded genetic algorithms have been employed for formulating the mesh partitioning algorithms. It is however appropriate to mention here that the binary GA can as well be employed for this formulation. However, as already discussed in section-2 of this paper, the float-encoded GA has distinct advantages over binary representation while searching continuous parameter spaces.

Although the genome representation in the proposed formulation is simple and convenient, the cost involved in evaluation of fitness is proportional to the size of the graph. For large size problems, it becomes computationally quite expensive. In order to improve the computational efficiency, several acceleration refinement techniques like adaptively shrinking the search space, switching over to interface refinement etc. have been attempted. Numerical studies indicate that FEGA with shrinking and interface refinement consistently gives good results. The shrinking of search space has minor impact in minimising the computational cost. However the interface refinement found to improve computational performance up to 14% for moderate size problems.

The micro-float-encoded genetic algorithms (μ-FEGA), which operate with a small population, have been developed and tried out for the mesh partitioning problem. They are computationally inexpensive. However, the quality of results obtained using μ-FEGA is marginally inferior when compared to the FEGA formulations. Further tuning of the GA parameters may improve the quality of partitions. The results presented in this paper (including visuals) demonstrate that the μ-FEGA based algorithms generate consistently better partitions than the multilevel spectral algorithm (RSBM). Similarly when compared with the multilevel graph partitioning algorithms like METIS, GA based partitions are better in most of the numerical examples solved and presented in Tab. 12. However, it is appropriate to mention here that the GA based algorithms including the fastest μ-FEGA algorithms are much slower than METIS and RSBM. In view of this, this method in the present state can be used for applications which consume substantial runtime on parallel processors (typically problems like nonlinear dynamics and computational fluid dynamics) so that the overheads associated with partitioning becomes negligible. These algorithms are certainly not practical for problems like parallel sparse solutions, which require partitions to be generated rapidly. However, we aim to investigate these techniques further to improve the computational performance by moving them on to parallel machines and also to improve the quality of submeshes by introducing some additional parameters like submesh aspect ratios.

Finally it should be mentioned here that the objective of the present work is not to suggest the proposed GA based algorithms as an alternative to popular mesh partitioning algorithms like METIS. The emphasis is only to demonstrate the potential of genetic algorithms in devising successful partitioning algorithms which are capable of generating partitions superior in quality when compared to the state-of-the-art mesh partitioning algorithms like METIS.

Acknowledgement: This paper is being published with the permission of Director, Structural Engineering Research Centre, Chennai. The authors would like to acknowledge the reviewer of the paper for his valuable suggestions to improve the quality of the paper.

References

Antonisse, J. (1989): A new interpretation of scheme notation that overturns the binary encoding constraint. In: J. David Schaffer(ed) *Proceedings of the third international conference on genetic algorithms*, Morgan Kaufmann Publishers, San Mateo, pp.86-91.

Atluri, S. N.; Han, Z. D.; Shen, S. (2003): Meshless Local Patrov-Galerkin (MLPG) approaches for weakly-singular traction & displacement boundary integral equations, *CMES: Computer Modeling in Engineering & Sciences*, vol. 4, no. 5, pp. 507-517.

Atluri, S. N.; Shen, S. (2002): The meshless local Petrov-Galerkin (MLPG) method: A simple & less-costly alternative to the finite element and boundary element methods. *CMES: Computer Modeling in Engineering & Sciences*, vol. 3, no. 1, pp. 11-52

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

Bouhmala, N.; Pahud, M. (1998): A parallel variant of simulated annealing for optimizing mesh partitions on workstations. *Advances in engineering software*, vol. 29, pp. 481-485

De Keyser, J.; 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*, vol. 28, pp. 579-602.

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

Giannakoglou. K. C.; Giotis. A. P. (2001): Unstructured 3-D Grid Partitioning Methods based on Genetic Algorithms, (Personal communication).

Gil, C.; Ortega, J.; Diaz, A. F.; Monotoya, M. G. (1998): Annealing based heuristics and genetic algorithms for circuit partitioning in parallel test generation. *Future generation computing systems*, vol.14, pp. 439-451.

Goldberg, D. E. (1989): *Genetic algorithms in search, optimisation and machine learning*, Addison-Wesley. N.J.

Goldberg, D. E. (1989): Sizing populations for serial and parallel genetic algorithms. In: *Proceeding of the third international conference on genetic algorithms. Morgan Kaufmann Publishers*. pp.70-79

Goldberg, D. E. (1991): Real–Coded Genetic Algorithms, Virtual Alphabets, and Blocking. *Complex Systems*. Vol. 3, pp. 153-171.

Hendrickson, B.; Leland, R. (1995): A Multilevel algorithm for partitioning graphs. *In proceedings of the supercomputing' 95. New York, NY 10036, ACM Press*.

Holland, J. (1975): *Adaptation in natural and Artificial Systems*. University of Michigan press. Ann Arbor.

Jones, M. T.; Plassmann, P. E. (1994): Computational results for parallel unstructured mesh computations. *Computing systems in Engineering*, vol. 5, pp. 297-309.

Kalyanmoy Deb.; Ram Bhushan Agarwal. (1995): Simulated Binary Crossover for Continuous search space. *Complex systems*, vol. 9, pp. 115-148.

Kalyanmoy Deb.; Goyal. M. (1996): A combined genetic adaptive search (GeneAs) for engineering design. *Computer Science and Informatics*, vol. 26, pp. 30-45.

Karypis, G.; Vipin Kumar. (1998): Multilevel k-way partitioning scheme for irregular graphs. *Journal of parallel and distributed computing*, vol. 48, pp. 96-129.

Karypis, G.; Vipin Kumar. (1999): A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of scientific computing*, vol. 20, pp. 359-392.

Kernighan, B. W.; Lin, S. (1970): An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, vol. 29, pp. 291-307.

Khan, A. I.; Topping, B. H. V. (1998): Subdomain generation for parallel finite element analysis. *Computing* *systems in engineering*, vol. 4 , pp. 96-129.

Koza, J. R. (1992): *Genetic Programming.* MIT Press. Cambridge.

Krishna Kumar, K. (1989): Micro genetic algorithms for stationary and non-stationary optimisation. *Intelligent control and adaptive systems*, SPIE Proceedings Volume 1196, Philadelphia.

Mansour, N.; Fox, G. C. (1991): A hybrid genetic algorithm for task allocation in multi-computers. In: R.K, Belew; L.B, Booker (ed), *Proceedings of the 4th International conference on genetic algorithms*, Morgan Kaufmann, pp. 466-473.

Mansour, N.; Fox, G. C. (1994): Allocating data to distributed memory multiprocessors by Genetic Algorithms. *Concurrency: practice and Experience*, vol. 6, pp. 485-504

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

Pain, C. C.; De Oliveira, C. R. E.; Goddard, A. J. H. (1999): A neural network graph partitioning procedure for grid based domain decomposition. *International Journal for Numerical Methods in Engineering*, vol. 44, pp. 593-613.

Park, K.; Carter, B. (1995): On the effectiveness of genetic search in combinatorial optimisation . In: *Proceedings of the 10th ACM symposium on Applied Computing, Genetic algorithms and Optimisation track.*

Plaskacz, E. J.; Ramirez, M. R.; Gupta, S. (1994): Non-linear explicit transient finite element analysis on the Intel Delta. *Computing systems in Engineering*, vol. 5, pp. 1-17.

Radcliffe, N. J. (1992): Non-linear Genetic Representations. In: Manderick, B.; Mannar, R.(ed) *Parallel Problem solving from Nature*, Elsevier Science Publishers, Amsterdam, pp. 259-268.

Rama Mohan Rao, A.; Appa Rao, T. V. S. R.; Dattaguru, B. (1998): Load balancing for parallel finite element analysis employing Artificial Neural Networks. In: P.K. Sinha (ed) *CDROM Proceedings of International conference on theoretical Applied Computational and Experimental Mechanics*.

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.; Dattaguru, B. (2001): A multilevel mesh partitioning technique employing genetic algorithms. In: Banishingh (ed) *Post conference proceedings of International conference on mathematical modelling* , pp. 371-375.

Simon, H.D. (1991): Partitioning of unstructured problems for parallel processing. *Computing systems in Engineering*, vol. 2, pp. 135-148.

Soper, A. J.; Walshaw, C.; Cross, M. (2001): A combined evolutionary search and multilevel optimisation approach to graph partitioning. *Mathematics Research Report 00/IM/58*, School of computing and mathematical Sciences, University of Greenwich, London, UK.

Walshaw, C.; Cross, M. (1999): Parallel optimisation algorithms for multilevel mesh partitioning. *Mathematics Research Report 99/IM/44,* School of computing and mathematical sciences, University of Greenwich, UK

Wendl, A. (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*, vol. 3, pp. 457-481.