**Computers and Concrete**, *Vol. 23, No. 5 (2019) 311-320*
DOI: https://doi.org/10.12989/cac.2019.23.5.311


# An optimized mesh partitioning in FEM based on element search technique

V. Shiralinezhad and H. Moslemi*

*Department of Civil Engineering, Shahed University, Persian Gulf Highway, Tehran, Iran*

*(Received December 6, 2018, Revised March 9, 2019, Accepted April 9, 2019)*

**Abstract.** The substructuring technique is one of the efficient methods for reducing computational effort and memory usage in the finite element method, especially in large-scale structures. Proper mesh partitioning plays a key role in the efficiency of the technique. In this study, new algorithms are proposed for mesh partitioning based on an element search technique. The computational cost function is optimized by aligning each element of the structure to a proper substructure. The genetic algorithm is employed to minimize the boundary nodes of the substructures. Since the boundary nodes have a vital performance on the mesh partitioning, different strategies are proposed for the few number of substructures and higher number ones. The mesh partitioning is optimized considering both computational and memory requirements. The efficiency and robustness of the proposed algorithms is demonstrated in numerous examples for different size of substructures.

**Keywords:** substructuring; finite element method; mesh partitioning; computational cost optimization; genetic algorithm

## 1. Introduction

The finite element method has become a widely used technique to numerically approximate the solutions of partial differential equations that arise in different real-world engineering problems. However, many professional finite element software packages fail to meet the demand in the computation speed and memory usage since meshes often have large numbers of elements in large and highly-complex problems. Partitioning an unstructured finite element mesh into a set of subdomains can reduce computational cost and memory usage since a large set of equations is decomposed to several smaller sets of equations. However, the path of mesh partitioning remains a topic of concern. In addition to balancing the computational cost between substructures, the boundary nodes should be minimized to reduce the boundary set of equation. These conflicting objectives make the mesh partitioning an *Np-complete* problem (Gary and Johnson 1979) and optimum solutions are practically intractable in a reasonable amount of time.

The first substructuring method was introduced by Przemieniecki (1963) to calculate the stresses and deflections in an aircraft structure and analyze different components separately. The recursive bisection (RB) algorithms are the most commonly used partitioning method where the mesh is divided into two pieces and these pieces are recursively divided to two pieces consecutively. Ansari *et al.* (2017) have represented a thorough survey of the general field of domain decomposition methods. Simon (1991) proposed a decomposition algorithm which is based on the computation of an eigenvector of the Laplacian matrix associated with the mesh and distribute the subdomains over a large number of processors in a MIMD machine with distributed memory. However it was shown that the RB methods may produce a partition that is very far away from the optimal one. A multilevel version of RB method was introduced by Bernard *et al.* (1995) contracting the grid by selecting successive maximal independent sets of vertices and connecting these vertices into a coarser graph, finding the eigenvectors efficiently, and interpolating eigenvectors from coarser to finer graphs. Yang and Hsieh (2002) proposed an iterative mesh partitioning optimization for parallel nonlinear dynamic finite element analysis in which the partitioning results are adjusted iteratively until the workloads among the substructures are balanced reasonably.

Another category of partitioning methods is non-spatial mesh partitioning algorithm which completely rely on the adjacency information of the mesh elements and devoid of geometrical structures and the partitioning. A greedy approach is proposed by Farhat and Lesoinne (1993) which bites into the mesh in order to construct every subdomain and proved that this algorithm is quite versatile. Pothen *et al.* (1990) considered an algebraic approach to compute the vertex separators for partitioning sparse matrices. However, this method is expensive since it requires the solution of a complex eigenvalue problem. Pan and Zhou (2013) applied the substructuring technique to simulate the crack extension. They gather the elements involving the crack as a substructure. The size of substructure expands when the crack propagates. They agglomerated the stiffness matrix to express additional freedoms by the freedoms of original mesh. A parallel stabilized finite element method is proposed by Hussain *et al.* (2013) which parallelize the system using message passing interface specification and employed it for darcy flow in distributed systems. They augment the conjugate gradient method for the solution of stabilized mixed finite element. Predari *et al.* (2017)

*Corresponding author, Ph.D.
E-mail: h.moslemi@shahed.ac.ir



ISSN: 1598-8198 (Print), 1598-818X (Online)

modeled additional constraints with fixed vertices by a direct k-way greedy graph growing partitioning that properly handles fixed vertices. A layer-by-layer partitioning of finite element meshes for multicore architecture was presented by Novikov *et al.* (2016) using a neighborhood criterion to partition the mesh into layers and combining them into blocks and assigning them into different parallel processors. Badia and Verdugo (2018) investigated the use of domain decomposition preconditioners for unfitted finite element methods such as extended finite element method defining the coarse degrees of freedom in the definition of the preconditioner.

Due to the NP nature of the mesh partitioning optimization, several metaheuristic approaches have been proposed to address this problem. Khan and Topping (1993) employed a genetic algorithm linked to a neural network predictive module for partitioning the coarse initial background mesh and showed that near optimal partitions for finer graded meshes may be obtained economically. Mohan Rao *et al.* (2002) employed genetic algorithm for mesh partitioning and used hierarchy of graphs to obtain the final graph partition. Float-encoded genetic algorithms are proposed by Kaveh and Bondarabady (2003) for mesh partitioning and several acceleration techniques like constraining the search space, local improvement after initial global partitioning have been attempted. Float-encoded genetic algorithm is proposed by Mohan Rao *et al.* (2004) in such a way that the number of variables considered in the chromosome construction was constant irrespective of the size of the problem. Colonies of artificial ant-like agents were employed by Korosec (2004) to restructure the resources in their environment in a manner which corresponds to a good solution of the underlying problem. Bahreininejad and Hesamfar (2006) applied the ant colony optimization for partitioning finite elements meshes using the swarm intelligence concept. They also improve the quality of the solutions by a recursive greedy algorithm optimization method. Mohan Rao (2008) presented an algorithm for generating sub-meshes with optimal shape using a steady state elite preserving evolutionary algorithm. In iterative algorithms the asperct ratio of finite element mesh is also important for faster convergence. Diekmann *et al.* (2000) designed the load balancing to maintain good partition aspect ratio and showed that cut size is not always the appropriate measure in load balancing. They presented a new center-based partitioning method of calculating the initial distribution which implicitly optimizes this measure. A new distributed multi-objective mesh-partitioning algorithm using evolutionary computing techniques was proposed by Mohan Rao (2009) to optimize the interprocessor communications and the submesh aspect ratios. A multilevel tabu search algorithm for balanced partitioning of unstructured grids proposed by Mehrdoost and Bahrainian(2016). They introduced A new tie-breaking strategy in selection of maximum gain vertices. Kaveh and Mahdavi (2015) used recently developed meta-heuristic algorithm, so-called Colliding Bodies Optimization (CBO) in conjunction with k-median method and compared it with standard Particle Swarm Optimization (PSO) to indicate that the CBO is capable of performing better decomposition using smaller or equal computational efforts. A hybrid ant colony together with genetic algorithm was employed by Kaveh and Shojaee (2008) for decomposing large-scale finite element meshes. Mohan Rao (2009) used the master-slave concept and proposed a new synchronous model to optimize the performance even on heterogeneous parallel hardware. Alternatively, a multiple population model was also developed which simulates its sequential counterpart. The advantage of the second model was that it could fit in large size problems with large population even on moderate capacity parallel computing nodes.

In recent years, several hopeful researches have been accomplished in the domain decomposition field. Marot *et al.* (2019) presented a new scalable parallelization scheme to generate the 3D Delaunay triangulation of a given set of points. They improved the Delaunay kernel to a multithreaded version that was able to concurrently insert vertices. A GPU domain decomposition solution for spectral stochastic finite element method was proposed by Stavroulakis *et al.* (2017) to address to address the intrusive stochastic mechanics problems. The solution of the resulting finite element algebraic equations was performed with the dual domain decomposition method, implementing specifically tailored preconditioners. Fu *et al.* (2017) proposed a novel partitioning method for block-structured adaptive meshes utilizing the meshless Lagrangian particle concept. They used high analogy of the problem to the relaxation of a multi-phase fluid to steady state. A new parallel domain decomposition algorithm based on integer linear programming (ILP) presented by Jordi *et al.* (2017) for the coastal ocean circulation models. Yui and Nishmura (2018) developed a cost effective graph-based partitioning algorithm and employed the nested dissection method to heuristically divide a system of linear equations, based on graph partitioning.

In this study, the mesh is partitioned via element search technique where each element is allocated to a substructure. In this manner, substructures may consist of some sparse elements which pass through the bounds of the neighboring substructures domain. The process of element allocation is accomplished by the genetic algorithm similar to the well-known hub allocation problem which has been applied to sensor networks, transportation systems, etc. The cost function in the optimization process can be regarded from two different aspects: the computational speed and memory usage. The most time-consuming parts of finite element method are iteratively solving linear systems derived from partial differential equation discretization. The most memory using parts of the method are the large coefficient matrices of these iterative solving linear systems. Thus, the focus of the cost functions is restricted to these decisive parameters. Since the number of internal and boundary nodes have different effects on the computational cost and memory usage, different partitioned meshes may be obtained via each of these cost functions. Thus a third combinatorial case of these two functions is defined to balance the computational speed and memory usage. In finite element meshes with millions of elements, the size of chromosome would increase drastically and GA algorithm

convergence would be disturbed. However, the proposed algorithm can afford problems up to 10,000 elements as shown in first example.

The remainder of this paper is organized as follows. Section 2 introduces the basic concepts of mesh partitioning theory relevant to our algorithm. Next, in Section 3, the optimization algorithm for mesh partitioning is presented with three cost functions: computational cost, memory usage and a combinational case of these two items. In each case, two different algorithms are introduced for the few number of substructures and higher number ones. Some numerical examples are presented in Section 4 to demonstrate the robustness and efficiency of the proposed mesh partitioning algorithms. Finally, several concluding remarks are given in Section 5.

## 2. Mesh partitioning

Przemieniecki (1963) first proposed substructuring method for first-level breakdown of complex systems for the displacement and force method. In this method, the whole structure is separated into smaller units called substructure. Each substructure consists of several elements and nodes. Each element corresponds to an exclusive substructure, but each node may be located at one or more substructures. From this point, the nodes of the FE model can be classified into two categories: The nodes that belong to an exclusive substructure called internal nodes and those which reside in two or more substructures called boundary nodes. This should be noted that global boundary nodes of the structure may be classified as internal node by this definition. The equilibrium equation for linear and nonlinear systems leads to an iteratively solving linear system which for the complete structure may be represented by

$$KU = F \tag{1}$$

This equation can be partitioned with the boundary displacements and forces separated from the interior ones as follows

$$\begin{bmatrix} [K]_{ii} & [K]_{ib} \\ [K]_{bi} & [K]_{bb} \end{bmatrix} \begin{Bmatrix} \{u_i\} \\ \{u_b\} \end{Bmatrix} = \begin{Bmatrix} \{F_i\} \\ \{F_b\} \end{Bmatrix} \tag{2}$$

where subscripts $i$ and $b$ denote the internal and boundary nodes, respectively. The displacement of internal nodes can be expressed in terms of displacements of boundary nodes. This elimination process is called static condensation. Solving Eq. (2) for $\{u\}_i$ and $\{u\}_b$ gives the internal and boundary degrees of freedom.

$$\{u\}_i = [K]_{ii}^{-1} \left( \{F\}_i - [K]_{ib} \{u\}_b \right) \tag{3}$$

$$\{u\}_b = \left[ \overline{K} \right]^{-1} \left\{ \overline{F} \right\} \tag{4}$$

where the assembled condensed stiffness matrix and condensed force vector are defined as

$$\left[ \overline{K} \right] = [K]_{bb} - [K]_{bi} [K]_{ii}^{-1} [K]_{ib} \tag{5}$$

$$\left\{ \overline{F} \right\} = \{F\}_b - [K]_{bi} [K]_{ii}^{-1} \{F\}_i \tag{6}$$

It is obvious that the linear solving system after condensation in Eqs. (3)-(4) is very smaller than linear solving system before condensation in Eq. (1). This effect increases the computation speed and reduces the memory usage. However, the efficiency of the technique depends on how the mesh is partitioned among the substructures. In next section, a set of algorithms based on element search technique are presented to optimally partition the mesh minimizing computational cost and memory usage.

## 3. Algorithms for partitioning

In this section some algorithms are presented to partition the mesh among substructures. The element search technique is used to assign each element to a substructure. The number of substructures is predefined in the problem and a permutation of these numbers can be assigned to each element. Different combinations of these assignments can be compared with respect to a cost function. Since the computation speed and memory usage are the main limiting parameters in a finite element analysis, these two parameters are considered in cost functions. Although a combinational case of these two cost functions is considered to optimize the mesh partitioning with respect to both parameters. Balancing the number of boundary nodes and internal node as defined in section 2, has a great effect in optimizing the cost functions. In problems with smaller number of substructures, the number of boundary nodes in initial population is small and it should be increased in iterations of the algorithm, but in problems with higher number of substructures the conditions are reversed and the number of boundary nodes should decreased iteratively. Thus, different strategies are proposed for mesh partitioning of the problems with a few number of substructures and higher number ones.

### *3.1 Optimization with respect to computational cost*

Different paths of substructuring may lead to iterative solving systems with various dimensions and consequently, mesh partitioning has a key role on the time of finite element analysis. Thus, the mesh partitioning can be controlled in such a way that minimizes the computational cost of finite element procedure. The most time-consuming parts of finite element method are iteratively solving linear systems derived from partial differential equation discretization and the cost function is defined with respect to these parts of analysis. It can be seen from Eqs. (5)-(6) that for each substructure, the dimension of solving linear equations is the number of internal degrees of freedom of that substructure. It is also obvious from Eq. (4) that a linear system should be solved with dimension of the total boundary degrees of freedom. The floating point operations needed for solving linear equation with $n$ unknowns, is in order of $n^3$. Thus, the computational cost function in a substructuring analysis process can be approximated by

$$Time\ \ Cost\ \ Function = N_b^3 + \sum_{i=1}^{NS} N_i^3 \tag{7}$$

where $N_b$ denotes the total number of boundary degrees of freedom, $NS$ indicates the number of substructures and $N_i$ denotes the number of internal degrees of freedom in each substructure. It is evident that this function is minimized if all of the parameters set to be equal. Thus, the internal nodes should be distributed equally between substructures and the total boundary nodes should have the same size too. These parameters have reverse relation and decreasing the internal nodes will lead to the increase of boundary nodes and vice versa. Therefore, several algorithms are proposed for finding the optimized case based on element search technique. Equalizing the number of the internal nodes is very simple, but the challenging task is the adjustment the size of the boundary nodes with them. The relation of boundary nodes and internal nodes of substructures is complicated and cannot be expressed mathematically. In the problems with small number of substructures, the initial number of boundary nodes is small and some algorithms are needed to increase them. In return, in problems with large number of substructures, the number of the boundary nodes is high initially and some strategies should be taken to reduce them. Thus, two different types of algorithms are proposed for these two cases. The choice of the appropriate case depends on the number of nodes and substructures and can be distinguished by comparing the number of boundary nodes and internal nodes in the initial partitioning state.

An appropriate initial population has a vital effect on the efficiency of the proposed genetic algorithm. In the process of the generation of the initial population, if the elements are distributed randomly between substructures, the number of the boundary nodes will be very high and the internal nodes will be negligible. Therefore the number of iterations in the genetic algorithm will increase drastically. The adjacency of the elements is utilized for the proper generation of the initial population. This means that first an element is selected randomly for each substructure, and then from each of these elements another adjacent element is added to the corresponding substructure. The adjacent element in 2D elements is defined as an element with two common nodes. In this way, the area of each substructure will grow and this process is repeated until all of the elements are located at one of the substructures. This algorithm avoids from the dispersion of the substructures and leads to the reduction of the boundary nodes. It should be noted that in the initial population, each member of the population is correspondent with a different mesh partitioning. The differentiation between the members is achieved by random selection of initial elements. The size of each chromosome is equal to the number of elements and the chromosome contains the substructure labels correspondent with each element. The size of the initial population depends on the number of substructures and would be increased accordingly.

After the generation of initial population, the number of boundary nodes is small and the genetic algorithm is employed to make the substructures sparse and balance the internal and boundary nodes. In first step the cost function is estimated for each member of the population according to Eq. (7) and the population is sorted with respect to the cost function. In next step, several parents are selected using roulette wheel, and crossover operation is accomplished. In this problem, the length of each chromosome is equal to the total number of the elements and each gene contains the substructure number of corresponding element. Uniform crossover is employed for generation of offsprings and each bit from the offspring's genome is independently chosen from the two parents according to a random distribution. Since individual genomes correspond to elements of the model and have no significant difference, uniform crossover is more proper than single-point or double-point crossovers. Thus, new generated mesh partitioning is a combination of the previous ones. In addition, mutation occurs during evolution according to a user-definable mutation probability. Since the number of internal and boundary nodes in initial population have a considerable difference, high probability of 10% is set in this study. In the mutation process, the content of some of the bits are changed randomly. This means that the substructure number of some elements is altered. For efficient of the mutation process, the elements will be renumbered which are located in the substructure with highest number of internal nodes to balance the internal nodes. Finally, individual genomes are

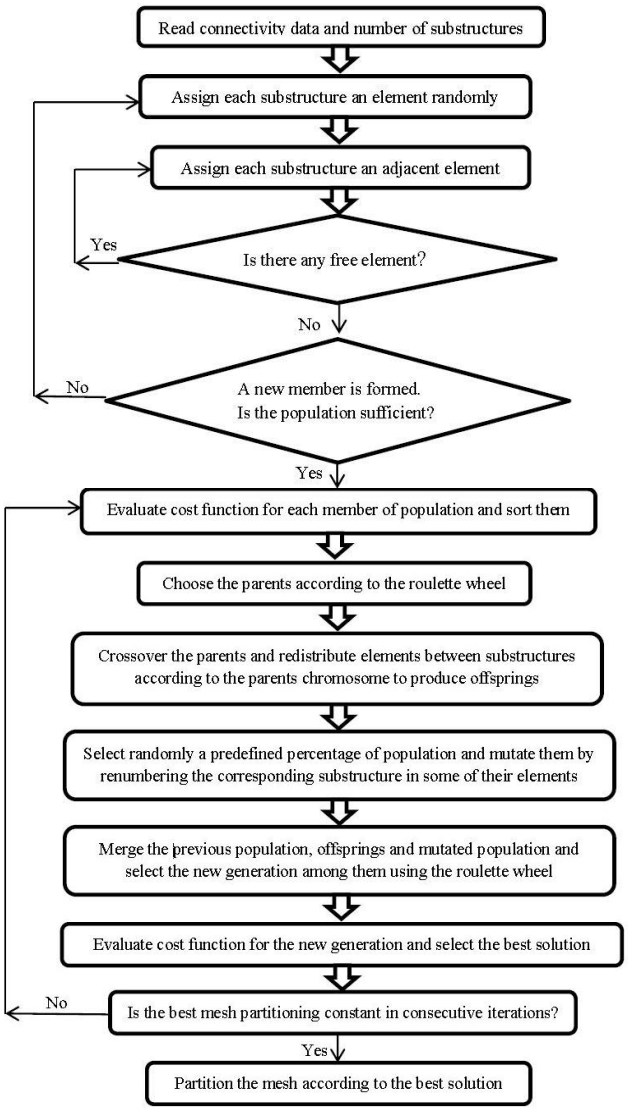



Fig. 1 Flowchart of the proposed genetic algorithm

chosen from the initial population, offsprings and mutated members during a roulette wheel selection process. The process of crossover and mutation is accomplished for new population repeatedly until the best mesh partitioning with minimum cost function is not changed in some predefined iterations. The entire flow of the proposed algorithm is shown in flowchart in Fig. 1.

If the problem consists of many substructures, the number of boundary nodes will exceed the internal nodes in that very initial population and the previous algorithm will lead to the increasing of the boundary nodes and consequently the cost function and analysis time. In this condition, the following algorithm will be applied. In this algorithm, a virtual master substructure is defined which distributes elements to the real substructures. First element assignment to the substructures is random, but in the next levels the elements are transferred from the master substructure to the real substructures considering the adjacency of elements to the substructures. Since several different elements may have adjacency conditions, that element will be transferred which produces fewer cost function according to Eq. (7).

It should be noted the cost function is evaluated for elements that are excluded from master substructure and the elements of the master substructure are ignored in cost function before transfer. The elements are transferred to the substructures continuously until there would be no adjacent element for the target substructure. Thus, the growth of a substructure may stop earlier than others which lead to the formation of unequal substructures. In the next step, the produced substructures are balanced by exchanging the elements. The substructure with minimum internal nodes is chosen and then the largest adjacent substructure is selected and the adjacent elements are transferred from larger substructure to smaller one. This process is repeated until the difference between internal nodes reach a predefined value. However, the balanced substructuring does not create optimum case necessarily. Thus, a final modification is accomplished to check the possibility of cost function reduction in exchange of the elements. The adjacency of element and substructure is the necessary condition in this step. Unlike the previous algorithm, the equality of internal and boundary nodes is not usually attained in this algorithm.

### 3.2 Optimization with respect to memory usage

In finite element analysis of large-scale structures, a common problem which users encounter is the insufficient required memory which denies the start of the analysis. One of the advantages of the mesh partitioning is the reduction of the dimension of matrices and gradual creation of them in the analysis process. In this way, the allocated memory can be released in next steps. One of the main ideas in substructuring is to partition the mesh in such a way that the minimum memory will be required in the analysis process. The main memory using parts of the algorithm are the large coefficient matrices of the iterative solving linear systems. Thus, the memory units needed in different steps of the substructuring procedure is computed primarily and then the partitioning is optimized with respect to this cost function. In computation of the cost function, the inactive memory is released whenever it is not required in next steps. Investigating the Eqs. (3)-(6) it can be seen that the maximum required memory is in the computation of the displacements of the boundary nodes.

The memory units needed for storing the coefficient matrices with n degrees of freedom, is in order of $n^2$. Thus, the memory units needed in the computation of boundary nodes displacements equivalent $N_b^2$ where $N_b$ denotes the total number of boundary degrees of freedom. In addition, in the process of solving the displacements of the internal nodes, required memory units are equivalent to $N_i^2$. However the displacements of internal nodes can be computed independently for each substructure and used memory in previous substructures can be released in each step. Thus, the maximum required memory in this part corresponds to the substructure with the most internal nodes which is denotes by $N_{i,cr}$. Consequently the memory cost function can be approximated by

$$Memory\ Cost\ Function = N_b^2 + N_{i,cr}^2 \tag{8}$$

Similar to the time optimization, there are two distinct algorithms for meshes with large number of substructures and those with smaller ones. The bounding limit of these two cases is the balance of the number of boundary nodes and internal nodes in the initial partitioning state. The algorithms presented in section 3.1 can be applied in this case too, but the memory cost function in Eq. (8) should be employed for comparison and sorting. In addition, in the balancing stage, the elements should be transferred from substructure with critical internal nodes $N_{i,cr}$.

### 3.3 Combinational optimization

In the two previous sections, the mesh partitioning algorithm was described on the basis of computational time and memory usage. Since there are different cost functions in these two cases, two distinctive mesh partitions can be achieved through each strategy for a certain structure. Each of these partitions has a shortcoming with respect to computational speed or memory usage. Thus, a third combinational case has been defined to have efficiency in both speed and memory. For this purpose, new cost function is formed from the combination of the time cost function and memory cost function as expressed in Eqs. (7)-(8). However, these two cost functions are two distinct quantities, one of them shows the memory units and another indicates the number of floating point operations. Therefore, to make them combinable, they have been normalized firstly. In the procedure of normalizing, an arbitrary uniform mesh partitioning is created for the specified structure and time cost function and memory cost function are computed for this partitioning according to Eqs. (7)-(8) labeled by $TCF_u$ and $MCF_u$. These two parameters are set as the reference costs for all of the probable partitions and cost functions are normalized with respect to these two parameters. For the combination of these two normalized cost functions, different weights can be assigned to each of these cost functions, depending whether speed or memory is more important factor in the analysis. Defining the weighting factor $r$, the combinational

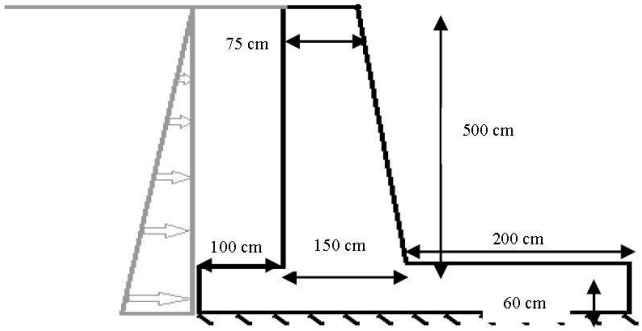



Fig. 2 The retaining wall; geometry and loading

cost function will be obtained by normalizing Eqs. (7)-(8)

$$r \times \frac{N_b^3 + \sum_{i=1}^{NS} N_i^3}{TCF_u} + (1-r) \times \frac{N_b^2 + N_{i,cr}^2}{MCF_u} \quad (9)$$

It is evident that $r$=1 corresponds to the computational optimization while $r$=0 coincide with the memory optimization. The mesh partitioning algorithm is similar to one described in section 3.1 for small and large number of substructures. In this algorithm the comparison between the various members of the population is accomplished considering combinational cost function given in Eq. (9).

## 4. Numerical examples

To demonstrate the capability and efficiency of the proposed optimization algorithm described in section 3, some examples are analyzed numerically. Three different examples are investigated where their mesh partitioning is optimized with respect to computational speed, memory usage and combinational case, respectively. To illustrate the generality of the algorithm, there are both small and large number of substructures in the examples. The complex geometry is selected for the examples to challenge the ability of the algorithm in such problems. All of the problems are meshed with two dimensional triangular elements. The mesh density is not uniform and is adapted to have a fine mesh in critical points of the problem. In all of the examples, half of the population is transferred to next generation using roulette wheel. The mutation is accomplished on 5% of population and in this mutated population, 10% of the bits of the genome have been changed. The termination condition of the iterations is the stability of the best solution in 10 consecutive iterations for all of the examples. The results of mesh partitioning and their costs are compared with those reported by Bahreininejad *et al.* (2006) which employed greedy algorithms for subdomain generation.

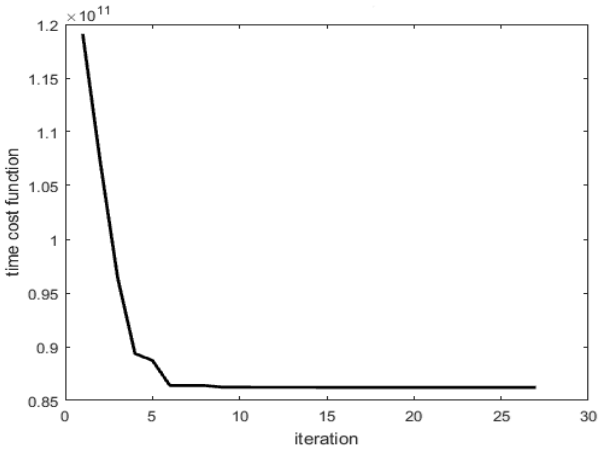



Fig. 3 The retaining wall; convergence rate of cost function

### *4.1 Optimization of the concrete retaining wall with respect to computational cost*

The first example is of a concrete retaining wall under soil pressure as shown in Fig. 2. The wall and its foundation is modeled and meshed with 10710 elements and 5749 nodes. This example is chosen to illustrate the ability of the proposed algorithm in problems with high number of elements. The model is analyzed with three substructures. In this example the mesh partitioning optimized with respect to computational speed. Since there are a lot of elements and a little substructures in the model, the internal nodes of each substructure overcome the total boundary nodes of the model (1769 vs. 147). Thus, the algorithm which proposed for small number of substructures would be applied.

As it was described section 3, in such conditions, the genetic algorithm will make the element distribution sparse to equalize the number of internal nodes of each substructure and total boundary nodes. The solutions have been converged after 27 iterations and the absolute minimum of the time cost function is attained. The element distribution in initial population and final mesh partitioning

Table 1 The retaining wall; Summary of element distribution in initial population and final mesh partitioning

| Number of nodes: 5749<br>Number of elements: 10710<br>Number of restrained nodes: 200 | | Substructure1 | Substructure2 | Substructure3 | Total boundary nodes | Number of iterations | Time Cost Function |
|---|---|---|---|---|---|---|---|
| Initial population | Internal nodes | 1769 | 1881 | 1752 | 147 | 27 | 1.91e11 |
| | Number of elements | 3579 | 3541 | 3590 | – | | |
| Final mesh partitioning | Internal nodes | 1388 | 1387 | 1387 | 1387 | | 8.62e10 |
| | Number of elements | 3582 | 3627 | 3501 | – | | |
| Greedy algorithm | Internal nodes | 1825 | 1791 | 1564 | 369 | - | 1.25e11 |
| | Number of elements | 3623 | 3557 | 3530 | - | | |

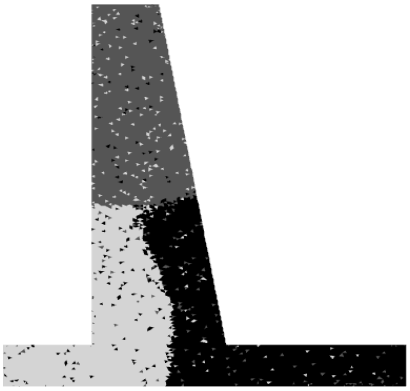


Fig. 4 The retaining wall; final mesh partitioning, each color corresponds to one of the substructures

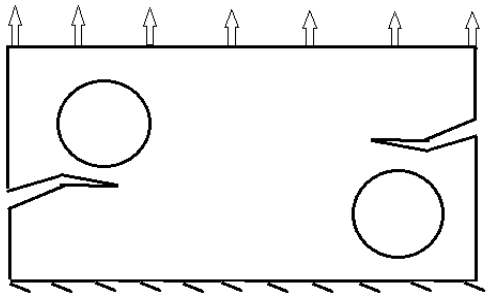


Fig. 5 Cracked rectangular plate; geometry and boundary conditions

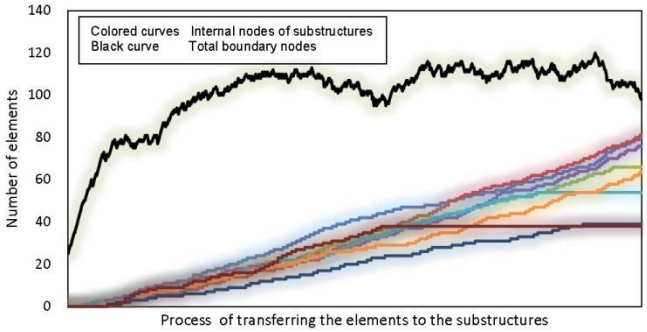



Fig. 6 Cracked rectangular plate; growth of the internal and boundary nodes in the process of the transferring of the elements

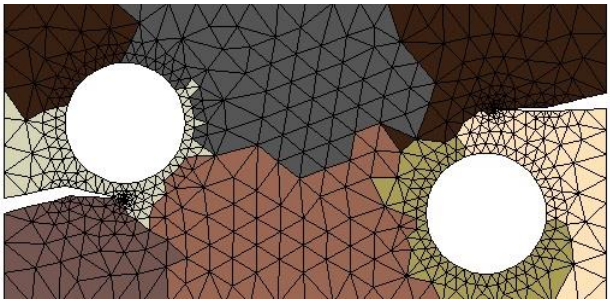


Fig. 7 Cracked rectangular plate; final mesh partitioning

and greedy algorithm is summarized in Table 1. It is obvious that the proposed algorithm have decreased time cost function more than greedy algorithm.

It can be seen that the cost function is decreased about 55% in the process of the proposed genetic algorithm. Eventual number of internal nodes of each substructure and total boundary nodes have been converged to 1387 which corresponds to absolute minimum of cost function. Downward trend of cost function in consecutive iterations are illustrated in Fig. 3. The high rate of convergence is obvious in this figure. The final mesh partitioning obtained after 27 iterations is sketched in Fig. 4. Each color indicates the elements distributed in one of the substructures. The sparse elements in the substructures are clear in this figure which cannot be seen in the classic mesh partitioning methods. It is due to the property of the proposed element search technique that permits the sparse elements to gather in a substructure for minimizing the cost function.

### *4.2 Optimization of the cracked rectangular plate with respect to memory cost*

The second example is a rectangular plate with two off-center holes which is subjected to a prescribed displacement in one edge as shown in Fig. 5. Two symmetric edge cracks are considered in the plate. The plate analysis is distributed over eight substructures and finite element discretization contains 619 nodes and 1033 elements. The memory is taken as the deciding factor for optimization in this example. The algorithm proposed for large number of substructures is applied in this example, because the number of the total boundary nodes exceeds the internal nodes in the initial population (103 vs. 81).

In this condition, the elements are transferred from a virtual master substructure to the real substructures. The process of accumulation of internal nodes of the substructures is illustrated in Fig. 6. It is obvious that the proposed algorithm have retained the balance between the substructures. However, the rate of the growth of the boundary nodes is higher than the internal nodes. Thus, the sparse elements are avoided in this model unlike the previous example. The final mesh partitioning of the model is presented in Fig. 7 which confirms the mentioned strategy.

Table 2 shows the number of internal nodes in eight substructures in the initial population and after convergence of the proposed algorithm and greedy algorithm. As it was described in Section 3 in large number of substructures, the equal substructures is not the optimum case necessarily and the final mesh partitioning has balanced the boundary nodes with internal nodes. It can be seen that the number of elements in substructure-8 is half of those in substructure-1. Balancing these values would lead to tremendous growth in boundary nodes as described in Section 3 in large number of substructures. This process has reduced the memory cost function from 133896 units to 33587 units (about 75% reduction in memory cost with respect to initial population and about 20% reduction with respect to greedy algorithm).

### *4.3 Combinational optimization of the knee lever*

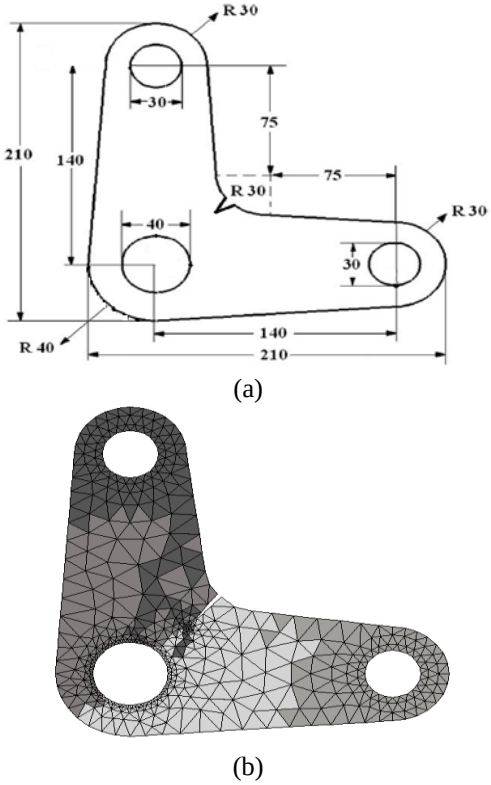
The last example presents a knee-lever which is notched in its middle part as shown in Fig. 8(a). This example is chosen to present a combinational optimization in a

Table 2 Cracked rectangular plate; Summary of element distribution in initial population and final mesh partitioning

| Number of nodes: 619<br>Number of elements: 1033<br>Number of restrained nodes:23 | | Sub1 | Sub2 | Sub3 | Sub4 | Sub5 | Sub6 | Sub7 | Sub8 | Total boundary nodes | Memory cost function |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial population | Internal nodes | 80 | 81 | 61 | 76 | 54 | 64 | 39 | 38 | 103 | 133896 |
| | Number of elements | 148 | 147 | 135 | 147 | 113 | 147 | 125 | 71 | - | |
| Final mesh partitioning | Internal nodes | 88 | 89 | 71 | 76 | 53 | 65 | 38 | 36 | 80 | 33587 |
| | Number of elements | 151 | 148 | 135 | 146 | 112 | 148 | 125 | 68 | - | |
| Greedy algorithm | Internal nodes | 82 | 71 | 65 | 71 | 85 | 40 | 58 | 32 | 92 | 42096 |
| | Number of elements | 148 | 160 | 146 | 145 | 153 | 72 | 136 | 73 | - | |

Table 3 The knee lever; Summary of element distribution in initial population and final mesh partitioning

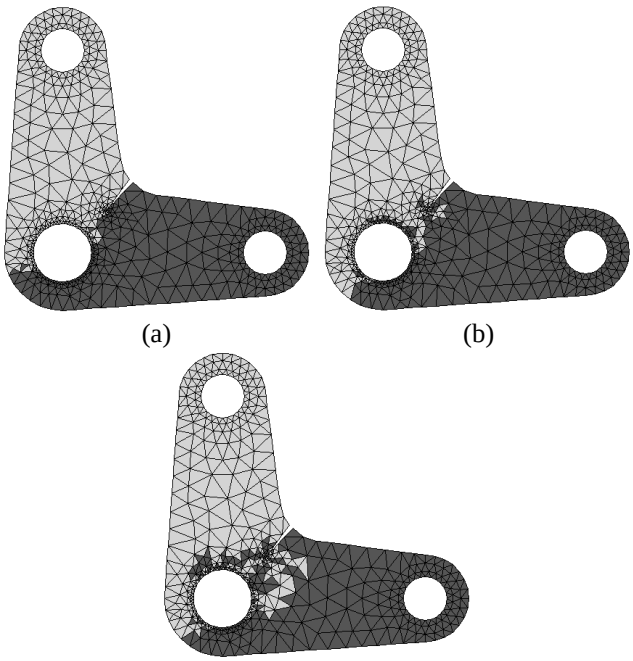
| Number of nodes: 550<br>Number of elements: 897<br>Number of restrained nodes: 8 | | Sub1 | Sub2 | Sub3 | Sub4 | Total boundary nodes | Memory cost function | Computational cost function | Combinational cost function |
|---|---|---|---|---|---|---|---|---|---|
| Initial population | Internal nodes | 94 | 147 | 129 | 131 | 41 | 26941 | 8.40e6 | 2.0 |
| | Number of elements | 154 | 251 | 240 | 252 | - | | | |
| Final mesh partitioning | Internal nodes | 95 | 136 | 129 | 93 | 89 | 22049 | 7.03e6 | 1.66 |
| | Number of elements | 195 | 241 | 241 | 220 | - | | | |




(a)

(b)

Fig. 8 The knee lever; (a) geometry (b) final mesh partitioning



(a) (b)

(c)

Fig. 9 The knee lever; combinational optimized mesh partition with different weighting factors (a) $r$=0 (b) $r$=0.5 (c) $r$=1

complicated geometry, which illustrates the capabilities of the proposed mesh partitioning procedure in handling complex structures. The importance of the computational speed and memory usage is considered to be equal in this example and consequently the weighting factor is set $r$=0.5. The structure is subdivided to four substructures. The finite element model has been implemented using 897 elements and 550 nodes. Since both memory and computation are present in cost function, an intermediate state of the two last examples has been achieved in this problem. The sparse elements are present in the substructures, but not as dense as the first example which was optimized with respect to the computational cost only. The final mesh partitioning is shown in Fig. 8(b). The computational and memory cost

functions of the initial population were considered for normalizing of the combinational cost function as was expressed by Eq. (9). The reduction of computational and memory cost functions via the proposed algorithm are given in Table 3. It is obvious that in the combinational optimization both memory and computational cost would decrease relatively (18% reduction in memory cost and 16% reduction in computational cost). The total combinational cost ratio has decreased from 2 to 1.66.

To illustrate the effect of the weighting factor on the combinational optimization, the last example is reanalyzed with two substructures and three different weighting factors $r$=0, $r$=0.5, $r$=1.0. The propagation of the elements to the opposite substructure will grow with the increase of the weighting factor as it is shown in Fig. 9.

## 5. Conclusions

In the present paper, an optimization algorithm was presented for the mesh partitioning in the finite element method. The proposed algorithm was based on the element search technique which allows sparse elements in the partitions unlike the classic algorithms. The genetic algorithm was employed in the process of optimization and different mesh partitions are taken as the population. The genetic algorithm procedures such as selection, crossover and mutation were modeled like the well-known hub allocation problem. The optimization cost function was investigated considering three different important factors: computational speed, memory usage and combinational case. The functions were approximated mathematically in each case. Since the balance between the internal nodes and the boundary nodes is a main target in this process, different strategies were proposed for structures which were subdivided to small number of substructures and those with large number of substructure. Finally, the efficiency and robustness of proposed optimization algorithm in mesh partitioning were presented by three numerical examples with complex geometries. The different mesh partitions were achieved in computational, memory and combinational optimization which illustrates the effect of the optimizing parameter.

## References


Ansari, S.U., Hussain, M., Mazhar, S., Manzoor, T., Siddiqui, K.J., Abid, M. and Jamal, H. (2017), "Mesh partitioning and efficient equation solving techniques by distributed finite element methods: A survey", *Arch. Comput. Meth. Eng.*, **26**(1), 1-16. https://doi.org/10.1007/s11831-017-9227-2.

Badia, S. and Verdugo, F. (2018), "Robust and scalable domain decomposition solvers for unfitted finite element methods", *J. Comput. Appl. Math.*, **344**, 740-759. https://doi.org/10.1016/j.cam.2017.09.034.

Bahreininejad, A. and Hesamfar, P. (2006), "Subdomain generation using emergent ant colony optimization", *Comput. Struct.*, **84**, 1719-1728. https://doi.org/10.1016/j.compstruc.2006.06.002.

Barnard, S.T., Pothen, A. and Simon, H. (1995), "A spectral algorithm for envelope reduction of sparse matrices", *Numer. Lin. Algebra Appl.*, **2**, 317-334. https://doi.org/10.1002/nla.1680020402.

Diekmann, R., Preis, R., Schlimbach, F. and Walshaw, C. (2000), "Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM", *Parallel Comput.*, **2**, 1555-1581. https://doi.org/10.1016/S0167-8191(00)00043-0.

Farhat, C. and Lesoinne, M. (1993), "Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics", *Int. J. Numer. Meth. Eng.*, **36**, 745-764. https://doi.org/10.1002/nme.1620360503.

Fu, L., Litvinov, S., Hu, X.Y. and Adams, N.A. (2017), "A novel partitioning method for block-structured adaptive meshes", *J. Comput. Phys.*, **341**, 447-473. https://doi.org/10.1016/j.jcp.2016.11.016

Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman W.H. and Company, NY.

Hussain, M., Abid, M., Ahmad, M. and Hussain, F. (2013), "A parallel 2D stabilized finite element method for darcy flow on distributed systems", *World Appl. Sci. J.*, **27**, 1119-1125. DOI: 10.5829/idosi.wasj.2013.27.09.15177.

Jordi, A., Georgas, N. and Blumberg, A. (2017), "A parallel domain decomposition algorithm for coastal ocean circulation models based on integer linear programming", *Ocean Dyn.*, **67**, 639-649. https://doi.org/10.1007/s10236-017-1049-0.

Kaveh, A. and Bondarabady, H.A.R. (2003), "A hybrid graph-genetic method for domain decomposition", *Finite Elem. Anal. Des.*, **39**, 1237-1247. https://doi.org/10.1016/S0168-874X(02)00192-0.

Kaveh, A. and Mahdavi, V.R. (2015), "Optimal domain decomposition using Colliding Bodies Optimization and k-median method", *Finite Elem. Anal. Des.*, **98**, 41-49. https://doi.org/10.1016/j.finel.2015.01.010.

Kaveh, A. and Shojaee, S. (2008), "Optimal domain decomposition via p-median methodology using ACO and hybrid ACGA", *Finite Elem. Anal. Des.*, **44**, 505-512. https://doi.org/10.1016/j.finel.2008.01.005.

Khan, A.I. and Topping, B.H.V. (1993), "Subdomain generation for parallel finite element analysis", *Comput. Syst. Eng.*, **4**, 473-488. https://doi.org/10.1016/0956-0521(93)90015-O.

Korošec, P., Šilc, J. and Robič, B. (2004), "Solving the mesh-partitioning problem with an ant-colony algorithm", *Parallel Comput.*, **30**, 785-801. https://doi.org/10.1016/j.parco.2003.12.016.

Marot, C., Pellerin, J. and Remacle, J.F. (2019), "One machine, one minute, three billion tetrahedra", *Int. J. Numer. Meth. Eng.*, **117**, 967-990. https://doi.org/10.1002/nme.5987.

Mehrdoost, Z. and Bahrainian, S.S. (2016), "A multilevel tabu search algorithm for balanced partitioning of unstructured grids", *Int. J. Numer. Meth. Eng.*, **105**, 678-692. https://doi.org/10.1002/nme.5003.

Mohan Rao, A.R. (2008), "A mesh partitioning algorithm for generation of shape optimized submeshes using evolutionary computing", *Pollack Periodica*, **3**, 91-103. https://doi.org/10.1556/Pollack.3.2008.3.8.

Mohan Rao, A.R. (2009), "Distributed evolutionary multi-objective mesh-partitioning algorithm for parallel finite element computations", *Comput. Struct.*, **87**, 1461-1473. https://doi.org/10.1016/j.compstruc.2009.05.006.

Mohan Rao, A.R. (2009), "Parallel mesh-partitioning algorithms for generating shape optimised partitions using evolutionary computing", *Adv. Eng. Softw.*, **40**, 141-157. https://doi.org/10.1016/j.advengsoft.2008.03.017.

Mohan Rao, A.R., Appa Rao, T.V.S.R. and Dattaguru, B. (2002), "Automatic decomposition of unstructured meshes employing genetic algorithms for parallel FEM computationss", *Struct. Eng. Mech.*, **14**, 625-647.

https://doi.org/10.12989/sem.2002.14.6.625.
Mohan Rao, A.R., Appa Rao, T.V.S.R. and Dattaguru, B. (2004), "Generating optimised partitions for parallel finite element computations employing float-encoded genetic algorithms", *Comput. Model. Eng. Sci.*, **5**, 213-234. https://doi.org/10.1007/978-3-319-55669-7_9.
Novikov, A., Piminova, N., Kopysov, S. and Sagdeeva, Y. (2016), "Layer-by-layer partitioning of finite element meshes for multicore architectures", *Commun. Comput. Inform. Sci.*, **687**, 106-117. https://doi.org/10.1007/978-3-319-55669-7_9.
Pan, Q. and Zhou, C. (2013), "A finite element sub- partition method for simulating crack extension independent to global mesh", *Acta Mechanica Solida Sinica*, **34**, 13-19.
Pothen, A., Simon, H.D. and Liou, K.P. (1990), "Partitioning sparse matrices with eigenvectors of graphs", *SIAM J. Matrix Anal. Appl.*, **11**, 430-452. https://doi.org/10.1137/0611030.
Predari, M., Esnard, A. and Roman, J. (2017), "Comparison of initial partitioning methods for multilevel direct k-way graph partitioning with fixed vertices", *Parallel Comput.*, **66**, 22-39. https://doi.org/10.1016/j.parco.2017.05.002.
Przemieniecki, J.S. (1963), "Matrix structural analysis of substructures", *AIAA J.*, **1**, 138-147. https://doi.org/10.2514/3.1483.
Simon, H.D. (1991), "Partitioning of unstructured problems for parallel processing", *Comput. Syst. Eng.*, **2**, 135-148. https://doi.org/10.1016/0956-0521(91)90014-V.
Stavroulakis, G., Giovanis, D.G., Papadopoulos, V. and Papadrakakis, M. (2017), "A GPU domain decomposition solution for spectral stochastic finite element method", *Comput. Meth. Appl. Mech. Eng.*, **327**, 392-410. https://doi.org/10.1016/j.cma.2017.08.042.
Yang, Y.S. and Hsieh, S.H. (2002), "Iterative mesh partitioning optimization for parallel nonlinear dynamic finite element analysis with direct substructuring", *Comput. Mech.*, **28**, 456-468. https://doi.org/10.1007/s00466-002-0310-6.
Yui, H. and Nishimura, S. (2018), "A cost effective graph-based partitioning algorithm for a system of linear equations", *Int. J. Comput. Sci. Eng.*, **16**, 181-190. https://doi.org/10.1504/IJCSE.2018.090440.

*CC*