9

Finite Element Mesh Decomposition Using Evolving Ant Colony Optimization

Ardeshir Bahreininejad
Tarbiat Modares University
Iran

1. Introduction

Combinatorial optimization problems arise in many areas of science and engineering. Unfortunately, due to the NP (non-polynomial) nature of these problems, the computations increase with the size of the problem (Bahreininejad & Topping, 1996; Topping & Bahreininejad, 1997).

Finite Elements (FE) mesh decomposition (partitioning) is a well known NP-hard optimization problem and is used to split a computationally expensive FE mesh into smaller subdomains (parts) for parallel FE analysis. Partitioning must be performed to ensure:

- Load balancing: for a mesh idealized using a single element type, then the number of elements in each partition must be the same, and
- Inter-processor communication: the partitions must be performed so that the number of nodes or edges shared between the subdomains is minimized to ensure that the minimum inter-processor communication during the subsequent parallel FE analysis is achieved (Topping & Bahreininejad, 1997; Topping & Khan, 1996).

Numerous methods have been used to decompose FE meshes (Farhat, 1988; Simon, 1991; Toping & Khan, 1996; Topping & Bahreininejad, 1997).

For automatic partitioning of FE meshes, Farhat (1988) proposed a domain decomposition method which is based on a greedy algorithm. The method provides FE mesh partitions in relatively short duration of time. The division of a mesh with respect to assigning a certain number of mesh elements to a mesh partition may be accomplished with simple arithmetic. In this method, the partitions are created sequentially from an overall FE mesh until the number of partitions become equal to the desired number.

Each FE element node is assigned a weight factor which is equal to the number of elements connecting to that particular node. The inner boundary of a partition is defined as the common boundary between two partitions. Two elements are considered to be adjacent if they share a vertex (node). The number of elements per partitions is determined by the total number of elements in the mesh, the number of different type of elements used in the mesh (triangular, quadrilateral, etc.), and the number of required partitions. In the case of a single type of elements, it is equal to the ratio between the total number of elements within the mesh and the number of required partitions (Farhat, 1988).

Although Farhat's method provides quick partitioning of FE meshes, the optimality of the mesh partitions with respect to the number of interfaces between adjacent partitions is not

Source: Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, Book edited by: Felix T. S. Chan and Manoj Kumar Tiwari, ISBN 978-3-902613-09-7, pp. 532, December 2007, Itech Education and Publishing, Vienna, Austria

guaranteed. In addition, the resulting partitions by this method are sensitive to the elements and node numbering of the FE mesh. Hence, for a given mesh topology, different solutions may be found if different node/element numbering of mesh elements are used.

Simon's method (1991) performs recursive bisection of FE mesh and uses eigenvector information to determine the partitions which have an equal number of elements on each side of the bisected mesh and have a minimum inner boundary. The rationale behind using recursive bisection instead of dividing the mesh into *N* number of partitions in a single step is based upon the following considerations:

- It is easier to bisect a graph rather than dividing it into more than two parts. The graph is bisected such that the requirements of load balancing and the minimization of the inner boundary nodes/edges between parts are effectively met for the two partitions.
- There is an aspect of parallelism within the recursive bisection. Initially a mesh may be divided into two parts and then each of these parts may be worked upon in parallel to form four partitions of the total mesh under consideration. Hence, the extent of parallelism that may be employed increase exponentially with the increase in the number of recursions.

The Simon's method provides efficient partitions, however the main drawback is the computational cost to reach a desired solution which increases nonlinearly with the increase in the size of the mesh.

The Subdomain Generation Methods (SGM) proposed by Topping and Khan (1996) and followed on by Topping and Bahreininejad (1997) presented a technique which incorporates an optimization algorithm (genetic algorithms or Hopfield-type neural network) and a trained multi-layered feedforward neural network based on the backpropagation algorithm (Rumelhart et al., 1986; Pao, 1989; Topping & Bahreininejad, 1997) to decompose FE meshes. The trained backpropagation neural network is used to estimate (predict) the number of elements which will be generated inside every individual element of the (initial) coarse mesh after mesh generation procedure is carried out. The estimated number of elements is incorporated into the optimization algorithm (module) to decompose a coarse FE domain rather than decomposing the fine mesh generated from the refinement of the initial mesh.

Ant Colony Optimization (ACO) is a type of algorithm that seeks to model the emergent behaviour observed in ant colonies and utilize this behaviour to solve combinatorial problems (Colorni, et al., 1991; Dorigo & Gambardella, 1997; Bonabeau, et al., 2000; Maniezzo & Carbonaro, 2001). This technique has been applied to several problems, most of which are graph related because the ant colony metaphor can be most easily applied to such types of problems.

A hybrid optimization approach is presented here to solve the FE mesh bisection problem. The algorithm incorporates several ACO features as well as local optimization techniques using a recursive bisection procedure. The algorithm was tested on a FE mesh with refined mesh sizes of 27155 triangular elements.

The chapter consists of an introduction to the ACO technique in Section 2. Section 3 describes how the ACO concept can be applied to FE mesh bisection. Local optimization techniques have been presented to improve the solution quality of the ACO for FE mesh bisection problem.

The predictive ACO bisection approach is described in Section 4 which uses a trained multi-layered feedforward neural network based on the backpropagation algorithm. The trained neural network is used to estimate the number of triangular elements that will be generated

after FE mesh generation (refinement) is carried out. Section 5 presents a recursive mesh bisection case study using the proposed hybrid ACO and neural networks mesh recursive bisection procedure. This FE mesh is also partitioned using a greedy FE mesh decomposing algorithm proposed by Farhat (Farhat, 1988). The comparison between the obtained results from the proposed hybrid ACO method and Farhat's greedy algorithm is presented. Finally conclusions are given in Section 6.

2. The Ant Colony Optimization Method

The ACO is a heuristic technique that seeks to imitate the behaviour of a colony of ants and their ability to collectively solve a problem. It has been observed (Colorni, et al., 1991; Dorigo & Gambardella, 1997; Bonabeau, et al., 2000; Maniezzo & Carbonaro, 2001) that a colony of ants is able to find the shortest path to a food source. As an ant moves and searches for food, it lays down a chemical substance called *pheromone* along its path. As the ant travels, it looks for pheromone trails on its path and prefers to follow trails with higher levels of pheromone deposits.

If there are two possible paths to reach a food source, as shown in Fig 1, an ant will lay the same amount of pheromone at each step regardless of the path chosen (minor evaporation of pheromone occurs during time). However, it will return to its starting point quicker when it takes the shorter path which contains more pheromone. It is then able to return to the food source to collect more food.

Thus, in an equal amount of time, the ant would lay a higher concentration of pheromone over its path if it takes the shorter path, since it would complete more trips in the given time. The pheromone is then used by other ants to determine the shortest path to find food as described in (Dorigo & Gambardella, 1997; Bonabeau, et al., 2000).

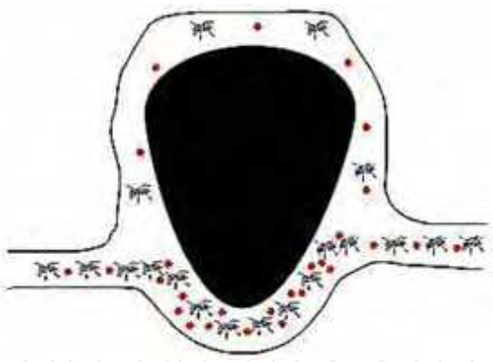


Figure 1. The pheromone deposition of ants (shown as dots) and their pursuing of the shortest path

The ACO technique has been successfully applied to the graph bisection problem by Bui and Strite (2002). They utilized the idea of finding shortest paths and the idea of territorial colonization and *swarm intelligence* in the ACO algorithms. Kuntz and Snyers (1994) and Kuntz, et al. (1997) applied these concepts to a graph clustering problem. Their algorithm combines the features of the ACO technique with swarm intelligence to form a model which is an artificial system designed to perform a certain task. Their model referred to the

organisms as *animats,* (ant agents), reflecting the fact that the system draws ideas from several sources and not just ant colonies. These ideas are important in the graph partitioning problem because the graph can be viewed as territory to be colonized (Bui & Strite, 2002; Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006).
The combination of these two ideas of animats following paths and forming colonies is used to solve the FE mesh bisection problem using triangular elements. However, the proposed ACO method can be used for the decomposition any graph based problem.

3. ACO for FE Mesh Bisection

The basic foundation of the ACO algorithm is to consider each (triangular) element in the FE mesh as a location that can hold any number of animats. The animats can move around the FE mesh by moving across (triangular) *edges* shared between two elements to reach a new element. Each animat belongs to one of two species (e.g. species A and B). However, animats of both species follow the same rules.
To start the algorithm, an initial number of animats are placed on the FE mesh. Their species and location are chosen randomly. At any point throughout the algorithm, the configuration of animats on the FE mesh constitutes a bisection of the mesh in the following way (Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006).
Each element is considered to be colonized by one species. At a given time, it is said to be colonized by whichever species that has the greater number of animats on it. Any ties are recorded and after the colonies of all other elements are calculated, the ties are broken in a random order by assigning the element to the species which results in a lower cut-size (inner boundary between bisections). The set of all elements colonized by species A constitutes A's colony and likewise the elements colonized by species B form B's colony.
In addition, each element can hold a quantity of pheromone. The two species produce separate types of pheromone, so an amount of A pheromone and/or B pheromone is left on each element. The idea of the algorithm is for each species of animats to form a colony consisting of a set of elements that are highly connected to each other while highly disconnected from the other colony. The result should be two sets of elements that are highly connected amongst themselves, but have few edges going between the two sets.
For an individual animat, the goal will be to lay down pheromone when the current element is a good position for animats of its own species and to move to new elements that it wants to add to its species' colony. If each animat follows these goals, the result will be a partitioning of the elements into two sets of similar size with few edges going between the two sets. A greedy algorithm is also formulated which fine tunes the bisection obtained from the ACO procedure.

3.1 ACO implementation

The ACO algorithm is an iterative procedure in which a percentage of animats are activated in each of the iterations. When an animat is activated, it adds an amount of pheromone to the element it is currently residing. It then may *die* with a certain probability or it may *reproduce* with a certain probability and it will move to a new element.
These operations involve only local information known by the animat. The animat is assumed to know the current time (i.e. iteration number), information about the element it is located at (such as the number and species of other animats on that element), and

information about the elements adjacent to its location. The mesh is updated with the new information after the completion every iteration.

The algorithm is divided up into S number of sets, each comprised of I number of iterations. After each set is carried out, the configuration of the mesh is forced into a bisection using a greedy algorithm (Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006). During each set, the parameters corresponding to the probabilities for *activation*, *death* and *reproduction* are modified. The parameters are changed in such a way that at the beginning of a set, colonial changes are high and by the end of the set the colonies should converged to a stable configuration.

The next set begins at the state where the previous set ended. However, if the animats follow their usual rules too soon, they may not be able to move away from the local optimum that has been reached.

Therefore, for all, but the initial set, a *shake* is performed for a certain number of the first iterations to help move the configuration, or distribution, of animats on the elements away from the solution to which it had prematurely converged as described in (Bui & Strite, 2002; Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006). The shake allows animats to select moves randomly instead of following the normal rules for movement. The *length of the shake* is changed during the algorithm. The first shake lasts for a fixed number of iterations and for the subsequent shakes, the length decreases linearly until the last set where no shaking occurs.

The bisection should come closer to the optimal bisection as the number of iterations increases and consequently shorter and shorter shakes are needed. The length of the shake is given by the following equation:

$$L_{shake} = \frac{L_{\max}(S_i - 2)}{2 - S_{total}} + L_{\max} \tag{1}$$

where L_{max} is the maximum shake length, S_i is the i^{th} set and S_{total} is the total number of sets.

3.1.1 Iteration and activation of animats

An iteration of the algorithm consists of a percentage of the animats being activated and then performing the necessary operations in parallel. The probability of an animat being activated changes during the set.

During the early iterations of a set, more animats are activated. By the end of a set, only a small percentage of the animats are activated in all iterations. The more animats that are activated in a single iteration, the larger the possible change in the configuration will be. The actual probability of activation is a sigmoid-like function given as:

$$a_{(I_I)} = \frac{1}{1 + e^{\frac{\ln(4)(2I_i - (I_{total}+1))}{I_{total}-1}}} \tag{2}$$

where I_i is the iteration number, I_{total} is the total number of iterations. To prove the Equation 2, consider the sigmoid function for the backpropagation algorithm activation function (Pao, 1989) given as:

$$a_{(I_I)} = \frac{1}{1 + e^{\frac{I_i - b}{\theta}}} \tag{3}$$

where b corresponds to the bias of the neural network and θ is the shape factor.
Considering the maximum (0.8) and minimum (0.2) values of the activation probability for the first and the I_{total} iterations respectively, chosen by the user and replacing them for $a_{(I_i)}$ in Equation 3, therefore:

$$\frac{b-1}{\theta} = \ln(4) \tag{4}$$

$$\frac{I_{total}-b}{\theta} = \ln(4) \tag{5}$$

Using Equations 4 and 5, b and θ are given by:

$$b = \frac{I_{total}+1}{2} \tag{6}$$

$$\theta = \frac{I_{total}-1}{2(\ln(4))} \tag{7}$$

Replacing Equations 6 and 7 in Equation 3, hence Equation 2 will be obtained. The function starts at a maximum and ends at a minimum. The maximum and minimum values are defined values set by the user in an ACO configuration file.
After the activations of animats have been completed, a percentage of the pheromone on each element is evaporated. Bui and Strite (2002) explain that the evaporation prevents pheromone from building up too much and highly populated elements from being overemphasized which in turn prevents the algorithm from converging prematurely. When an animat is activated:

- It deposits pheromone on its current element,
- It dies or reproduces with a certain probability,
- It moves to another element.

These operations are performed by the animat using local information to make decisions.

3.1.2 Pheromone

The purpose of pheromone is to allow the algorithm to retain a *memory* of good configurations that have been found in the past. Members of each species deposit their pheromone on an element to indicate that this is a good configuration and more animats of their species should come to this element.
The effect of pheromone on the overall performance of the algorithm is to control the animats' movements prohibiting them to set astray over the optimization domain, In other words, pheromone is used a means to control animats' movements throughout the algorithm, thus enabling the optimization process to move towards a solution.
When an animat is activated, it determines the colonization percentage of its adjacent elements. If the FE mesh element is highly connected to elements colonized by the animat's species, then the animat knows that this element is a good candidate for being colonized by its own species. The animat then attempts to reinforce this element by depositing a larger amount of its pheromone.
However, if the animat determines that the element is not highly connected to elements colonized by its own species, it will lay down less pheromone to discourage more of its

species to come to this element. In addition, the animats place lesser amounts of pheromone in the early iterations and more pheromone in later iterations.
The reason for this is that, in early iterations more change is needed. This allows the animats to explore more of the search space in the beginning and to exploit more of their current configuration near the end.
There is also a limit to the amount of pheromone of each species that can be stored on an element. The limit for the amount of pheromone for an element is the product of the connectivity degree of that element to its adjacent elements and the pheromone limit parameter. This allows densely connected elements to accumulate more pheromone. The formula for the amount of pheromone to be deposited is given as:

$$ph(a, I_i) = \frac{a_{col}}{a_{total}} \frac{I_i}{I_{total}} \tag{8}$$

where a is the animat, a_{col} is the number of elements adjacent to the animat's current location which are colonized by the animat's species, and a_{total} is the total number of elements adjacent to the animat's current location.

3.1.3 Death

The animat will die with a death probability which is fixed throughout the algorithm. The main purpose of death is to avoid overpopulation of animats. Overpopulation may influence the speed of computations. In addition, death can manipulate the algorithm's configurations by adding changes to animat's species.
The activation probability changes throughout the set so that early in a set, more animats are activated and therefore more animats die early in the set. The purpose of this is to have shorter life spans in the beginning, which allows more changes in the configuration. Later in the set, the animats are allowed to live longer and thus, there is less change and the solution is able to converge. An animat is removed from the list if it is selected to die.

3.1.4 Reproduction

If an animat is not selected for death, the algorithm proceeds to the reproduction step. An animat is selected for reproduction with fixed reproduction probability. However, the number of new animats that are produced depends on time (iterations).
In the first iteration of a set, the average number of animats born is β_{init} and it decreases linearly over time to β_{final} in the last iteration of a set. The changing birth rate serves to allow more change in earlier iterations, in which animats live for shorter lengths of time. The number of animats born is defined by:

$$\beta = \frac{-(\beta_{final} - \beta_{init})(I_i - 1)}{(1 - I_{total})} + \beta_{init} \tag{9}$$

The actual number of animats born is selected uniformly at random over a range, centered on the average birth rate for the iteration. The number of animats born can be up to β_{range} more or less than the specified average. The β_{range} is usually taken as 50 percent (Pao, 1989, Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006).
New animats are the same species as their parents. If the element on which the parent is located is colonized by its own species, the newly born animats are all placed on that

element. However, if the element is colonized by the opposite species, only β_{stay} percent of the offspring animats are placed there. The remaining new animats will be placed on the element to which the parent animat moves in the next step.

The justification for this is that, if the parent animat is already at an element populated by its own colony and moves to another element, it should leave its offspring behind to help maintain the majority on that element. If the parent's species is not in majority, it should take most of its children to the new element in which it is trying to create a colony. However, the parent leaves some of its offspring behind so that some of its species remain at that element (in case that element really should be part of their colony).

There are two other constraints on reproduction. First, there is a limit of β_{limit} to how many offspring an animat can produce during its lifetime. This value is fixed throughout the algorithm and is the same for each animat. Once the limit is reached, the animat can no longer reproduce. This serves to prevent one species from taking over the entire FE mesh and forcing the other species into extinction.

Another problem arises when an animat reproduces and places all of its children on its current element. Once one of the children is activated, it will in turn reproduce and deposit more children on the same element before moving.

This overemphasizes that element and does not allow the colonies to change much from their original starting configuration. Because of this, animats are not able to reproduce until they have made a fixed minimum number of moves. This ensures that the mesh is explored and that new configurations are created by the reproduction and movement rather than being inhibited by these operations.

Therefore, the main aim of reproduction is to encourage the species of animats which have colonized an element. This can influence the rate of convergence of the overall algorithm.

3.1.5 Movement

Movement is by far the most important operation the animats perform. The animats' movement is the main mechanism by which the solution is produced. The animat can move to any element which is connected to its current location by an edge. There are two factors used to select a move from the set of possible moves. For each element to which the animat could move, the connectivity to other elements is examined. The animat should move to an element that is highly connected to other elements colonized by its own species. In addition, the animat should learn from the past and take into account the pheromone that other animats have deposited.

Throughout the course of a set, these two factors are weighted differently. Initially, the pheromone is weighted at ω_{pmin} with the weight increasing linearly to ω_{pmax}. Conversely, the connectivity is weighted at ω_{cmax} to begin and decreases linearly to ω_{cmin}. In this way, the configuration of the colonies changes greatly in early iterations and over time, learning is incorporated into the algorithm.

These basic factors drive the animats to create colonies of highly connected elements which are highly disconnected from the elements colonized by the opposing species. These factors are the basis of move selection.

The probability of moving to an adjacent element is proportional to the two combined factors. Specifically, the factors are combined according to Equation 10 to create a probability of moving to a specific element e.

$$pr(e) = ce_c + pe_p + \varepsilon \tag{10}$$

where e_c is the number of elements adjacent to e that are colonized by the animat's own species, c is the connectivity weight where $\omega_{c\min} \le c \le \omega_{c\max}$, e_p is the amount of pheromone of the animat's species on element e, p is the pheromone weight where $\omega_{p\min} \le p \le \omega_{p\max}$, and ε is a fixed amount added to prevent any probabilities from being zero. The values of c and p are given by the following equations:

$$c = \frac{(\omega_{c\max} - \omega_{c\min})(I_i - 1)}{1 - I_{total}} + \omega_{c\max} \tag{11}$$

$$p = \frac{(\omega_{p\max} - \omega_{p\min})(I_i - 1)}{1 - I_{total}} + \omega_{p\max} \tag{12}$$

In order to encourage animats to explore more of the FE mesh, the probability of selecting the move which would result in the animat returning to its previous location is reduced. A factor is used and initialized by a defined value and decreases linearly after the completion of each set until it reaches zero in the final set.

Therefore, animats move in order to find suitable locations to colonize their species. The movement is aimed to find locations (elements) for colonization so that such locations are highly connected while highly disconnected from the locations of the colonization of other animat's species. This is the main purpose of the ACO algorithm for partitioning graphs.

3.1.6 Between the sets

After each set of iterations, several other operations are performed based on the history of animates activities.

First, the algorithm looks for *mistakes* the animats have made. The algorithm looks for individual elements with a high percentage of their adjacent elements colonized by the opposite species. Therefore an element colonized by spices A having a high percentage of it adjacent elements colonized by species B is swapped to the B colony.

Next the algorithm looks for any *discontinuities* which may be generated during each set. In each set the program swaps the smallest discontinuous colonization of each species. This is carried out using a recursive greedy optimization procedure (Bahreininejad, 2004; Bahreininejad & Hesamfar, 2006) which will be discussed further. As was discussed earlier, any given configuration of animats on the FE mesh does not necessarily induce bisection. Therefore, if one species is colonizing more elements than the other, some elements will be swapped to the other species. The elements to be swapped are selected from the set of *border elements*, that is, elements that are adjacent to an element of the opposite colony. By changing the colony of only border elements, the algorithm continues in the direction the animats were heading, rather than selecting elements in a region that is completely dominated by one species and creating an irregularity. Elements are selected to be swapped by making the greedy choice from amongst the border elements. This is carried out, for triangular elements, according to the pseudo code procedure shown in Table 1.

After each element is selected, the swap is performed and the subsequent choices are made based on the new configuration of the colonies. At this point, the two colonies form bisection and this information is recorded.

```
WHILE
    Number of elements in colony A is not equal to number of elements in colony B
DO
    BEGIN
      Swap border elements of colony with greater number of elements based on
      priority rules
      Priority Rules
        First, randomly swap elements with two edges in boundary and will result in
        one edge in boundary after swapping
        Second, randomly swap elements with one edge in boundary and will result
        in one edge in boundary after swapping
        Third, randomly swap elements with one edge in boundary and will result in
        two edges in boundary after swapping
END
```

Table 1. The pseudo rules for the element swap procedure

Now, if this was not the final set, the mesh and population is prepared to start a new set by performing two more manipulations. The number of animats on the mesh may differ from the initial number of animats of both species. Usually after a set, the number of animats is higher than the initial number. The problem with this is that, if it continues, the number of animats grows so large that the computations become prohibitively expensive (since a percentage of animats are activated in each of the iterations). To correct this, the number of animats is reduced to the initial number. This is carried out by randomly removing animats until the correct population size is reached. This disruption of the colonies is negligible since each new set begins with a shake anyway.

Finally, the number of animats in the two species is equalized. Normally the number of animats in each species is quite close, since the colonies have been forced into bisection. However, this may not always be the case. The bisection may not guarantee that the two species have the same number of animats. To improve this possible problem, animats are added to equalize the number of animats in each species. Usually this is a very small number and thus is not problematic in consideration of the previous operation (reducing the number of animats to the initial number). The new animats are added only to elements where their own species is already in majority. Thus, this operation does not significantly alter the configuration of the colonies; it merely gives added strength to the colonies in which animats are added.

Following this operation, a new set is begun. Again, the time (iteration) is initialized and all probabilities relating to time are reset. Therefore, as the animats have converged on a possible solution, starting a new set allows the animats to move away from that solution in expectation of finding a better solution in case this solution was a local optimum. After a total of specified sets have been completed, the solution should represent partitioning with minimum cut-size.

3.2 Greedy algorithm for partition enhancement

The ACO algorithm should ensure the minimum cut-size and balanced partitions. However, sometimes discontinuous partitions may be developed where *islands* of partitions are generated (e.g. a domain is bisected and three partitions are generated where one of the

partitions is composed of two separate partitions). A recursive greedy algorithm is used to improve the partition solutions given by the ACO and will swap the smallest generated discontinuity in each set.

The algorithm consists of two major parts. Initially, it searches and identifies each discontinuity. This is carried out by means of a recursive procedure which calls itself in order to give an index to an element and all its adjacent elements with similar spices.

Therefore, when an element colonized by one of the spices has been given an index, all its adjacent elements with the same species will also get the same index. This scheme is illustrated according to the pseudo code procedure shown in Table 2. The procedure in Table 2 is used by another procedure to index all the mesh elements using different index for each partition. This scheme is shown in Table 3.

After that the algorithm will swap the colonization of the smallest generated discontinuity with the colonization of other species in each set. This is carried out according to the pseudo code procedure shown in Table 3. Table 4 presents the pseudo code procedure for the proposed ACO-based algorithm for FE mesh recursive bisection.

3.3 Flying ants

Another approach to deal with cases where discontinuous partitions may occur was to assume that ants (animats) are able to fly which agrees with flying ants present in nature. This is especially true when a colony of ants may become localized and surrounded by ants of other colony and ants in the localized colony will find it impossible to search for better places to colonize.

```
Procedure GiveIndex(i)
BEGIN
    index of element i = index
    FOR k = 1 to Number of adjacent
      elements of element i
    IF colonization of adjacent element k = colonization of element i
    THEN
      GiveIndex(adjacent element k)
END
```

Table 2. The pseudo code for indexing an element of a discontinuity and all its adjacent elements with the same colonization

```
FOR I = 1 to NumElements do
  IF index of Element i = 0
  THEN
    BEGIN
      index = index + 1
      GiveIndex(i)
    END
Determine the index which refers to a discontinuity with smallest number of elements
in each species.
Swap the colonization of elements correspond to that index.
```

Table 3. The pseudo rule for indexing all the mesh elements using different indexing scheme and swapping process of the smallest discontinuity

This brings the idea of using flying ants approach to *prevent* ants getting stuck in localizations. In this approach, all the animats are capable of flying from the beginning of the optimization. The moving probability is determined using Equation (3) except that all the elements in the mesh are considered.
The animats can fly to the element with the highest moving probability in the mesh. However, the animats which already exist in the element with the highest moving probability prior to the arrival of new animats will have to move to the element with the second highest moving probability. This process is carried out by both spices of animats. The animats continue searching for better elements to colonize.

4. Neural Network Predictor

The Subdomain Generation Methods (SGM) proposed by Topping and Khan (1996) and followed on by Topping and Bahreininejad (1997) presented a technique which incorporates an optimization algorithm (genetic algorithms or Hopfield-type neural network) and a trained multi-layered feedforward neural network based on backpropagation algorithm. The trained backpropagation neural network is used to estimate the number of elements which will be generated inside every individual element of the coarse mesh after mesh generation procedure is carried out.
The estimated number of elements is incorporated into the optimization algorithm to decompose a coarse FE domain rather than decomposing the fine mesh generated from the refinement of the initial mesh.
A backpropagation-based multi-layered network with 5-12-8-6-1 (five units in intput layer, 12 units in the first hidden layer, 8 units in the second hidden layer, 6 units in the third hidden layer and finally one unit in the output layer) topology was adopted and trained which is capable of estimating a number up to 1760 triangular elements corresponding to the generated elements after mesh refinement is carried out.
The inputs to this network are the three scaled side lengths of each triangular element and the two scaled mesh parameters of each element. The scaling was made using one of the three mesh parameters (Topping & Bahreininejad, 1997).
The predicted number of elements generated in each element of the coarse mesh after mesh refinement, is used after the last set is completed. This information is used in a greedy algorithm which forces the solution to a bisection considering the same priority rules presented in Section 3.2.
The original SGM method partitions a FE mesh based on the coarse mesh using genetic algorithms (Topping & Khan, 1996). The generated subdomains are then refined individually using adaptive mesh refinement procedure.
In most cases, it has been observed that the total number of elements generated by the refinement of individual subdomains may not be the same as the total number of elements generated after the mesh refinement of the initial coarse mesh.
The presented method partitions the coarse mesh based on ACO and the SGM neural predictor approach and the resulting subdomains are mapped onto the final refined mesh.

```
BEGIN
 Randomly add animats to the mesh
 FOR Set = 1 to S
 BEGIN
  FOR iteration = 1 to I
  BEGIN
   FOR animat = 1to N
   BEGIN
    Activate e(iteration)% of animats
    Determine degree of species in each element
    IF animat activated
    BEGIN
     Add ph(animat, iteration) pheromone to animat's location
     Kill r_d% of animats of elements
     IF animat is not chosen for death
     BEGIN
      IF animat meets reproduction criteria
      BEGIN
       r_r% of animats reproduce
      ENDIF
      IF iteration is in shake length
      BEGIN
        Move animats randomly
      ELSE
         Move based on pheromone and connectivity
      ENDIF
     ENDIF
    ENDIF
   ENDIF
  ENDFOR
  Evaporate Y% of pheromone
 ENDFOR
 Look for mistakes and swap elements
 Run the greedy algorithm to reduce discontinuities
 Record the bisection solution
 Reduce total number of animats to the initial value
 Equalize the number of animats in each species
ENDFOR
Return the best solution
END
```

Table 4. The pseudo rules for the proposed ACO algorithm for FE mesh decomposition

5. ACO Partitioning Case Study

A case study is presented to illustrate the ACO-based optimization approach for recursive FE mesh bisection. The optimization procedure is based on flying ants ACO using the neural

network predictor. This method was compared with a greedy algorithm for partitioning FE meshes by Farhat.

The case study was carried out on a PC with Intel Pentium III 500 MHz processor. Twenty simulation runs were conducted and Table 5 represents the ACO parameters adopted for the final solutions. The computation times were less than 20 seconds for the proposed method and 38 seconds for the Farhat's method.

In this case study, an inverted U-shaped domain shown in Fig 2 was used for partitioning the domain into eight subdomains.

Fig 3 represents the resulting partitioned mesh by recursively bisecting the coarse mesh into eight subdomains using the proposed method. Fig 4 represents the partitioning of the refined mesh into eight subdomains using Farhat's method.

Table 6 shows the cut-size and the number of elements in each partition after recursive bisection using the proposed approach.

Table 7 shows the result obtained from the mesh decomposition into eight subdomains using Farhat's method.

The imbalance between the actual and the desired number of elements in each generated subdomain using the ACO method is shown in Table 6.

Parameter	**Value**
Number of iterations per set	100
Number of sets	10
Maximum shake length	5
Initial number of animats	100
Number of moves needed before an animat can reproduce	2
Maximum number of offspring per animat	10
Average number of animats born in first iteration	4
Expected number of animats born in final iteration	2
Minimum pheromone weight	0
Maximum pheromone weight	1
Minimum connection weight	250
Maximum connection weight	500
Pheromone limit	1000
Percentage range from average number of animats born	0.5
Maximum activation probability	0.8
Minimum activation probability	0.2
Death probability	0.035
Reproduction probability	0.011
Minimum probability for moving to an element	0.1
Reduction factor for returning to previous location	0.9
Percentage of offspring that stay on old location when not colonized	0.2
Percentage of adjacent elements needed for swap	0.75
Percentage of animats needed for majority	0.9
Evaporation rate	0.2

Table 5. Ant colony optimization run-time parameters

The load imbalance problem may have occurred from the inaccuracy of the predictive neural network to closely estimate the number of elements which will be generated in a single element of the coarse mesh after mesh refinement is carried out. A better trained neural network may improve the quality of the solutions.

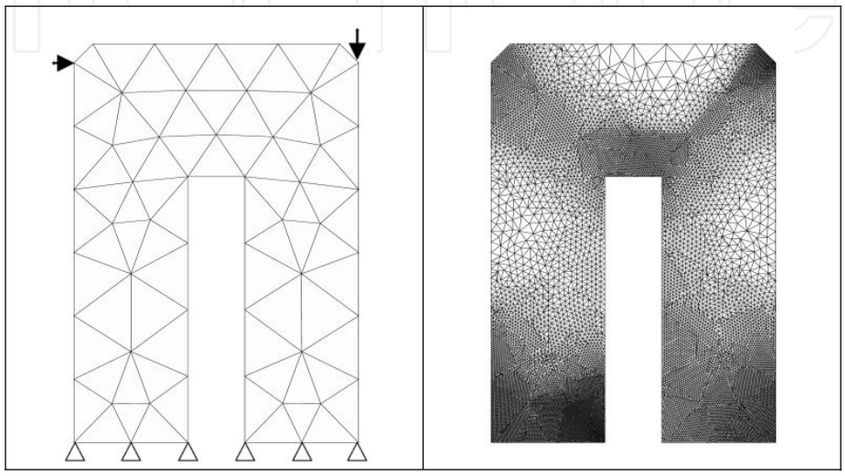


Figure 2. The initial mesh with 75 elements and the final refined mesh with 27155 elements

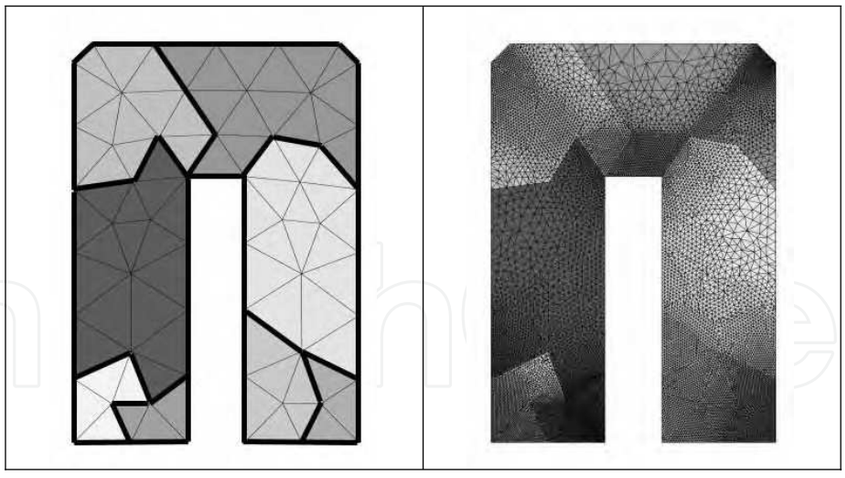


Figure 3. The initial coarse mesh and the final mesh devided into 8 subdomains using the proposed method

The imbalance of elements shown in Table 6 may not pose a serious threat. In fact, this may be used advantageously on coarse grained parallel networked computers where system

architecture and the computational load among machines may differ (heterogeneous environment).

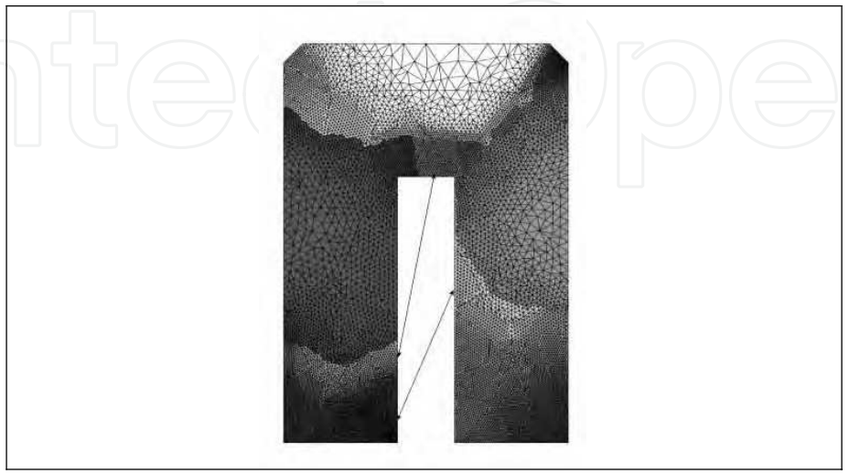


Figure 4. The the final mesh devided into 8 subdomains using the Farhat's method where the disjointed subdomains are shown with the arrows

On the other hand, Farhat's method offers the best load balancing results while producing a large number of interfacing edges. In parallel FE computations, the number of interfacing edges imposes a great deal of inter-processor communications. Thus the efficiency of parallel computations can greatly depend of lower cut-size (interfacing edges/nodes) between the subdomains. The proposed ACO method can produce much more appealing cut-sizes which reduces the inter-processor communications during parallel FE computations while producing satisfactory load balancing between partitions.

Subd. Number	**Num. of elements**	**Desired num. of elements**	**Diff.**
1	3266	3394.375	-128.375
2	3907	3394.375	512.625
3	3604	3394.375	209.625
4	3243	3394.375	-151.375
5	3881	3394.375	486.625
6	3400	3394.375	5.625
7	3153	3394.375	-241.375
8	2701	3394.375	-693.375
Total number of interfacing edges			370

Table 6. Comparison between the desired and the obtained number of elements in each subdomain and the number of interfacing edges using the proposed method

Subd. Number	Num. of elements	Desired num. of elements	Diff.
1	3395	3394.375	0.625
2	3395	3394.375	0.625
3	3395	3394.375	0.625
4	3395	3394.375	0.625
5	3395	3394.375	0.625
6	3395	3394.375	0.625
7	3395	3394.375	0.6255
8	3390	3394.375	-4.375
Total number of interfacing edges			1234

Table 7. Comparison between the desired and the obtained number of elements in each subdomain and the number of interfacing edges using Farhat's method

6. Conclusions

The application of ant colony optimization using swarm intelligence concepts, in combination with a trained feedforward neural network predictor which estimates the number of elements which will be generated within each element of the (initial) coarse mesh after mesh refinement is carried out, to the recursive bisection of finite elements meshes was described. This algorithm combines the features of the classical ant colony optimization technique with swarm intelligence to form a model which is an artificial system designed to perform a certain task. This model is used to solve the finite elements mesh recursive bisection problem which should ensure the minimum cut-size between bisections while maintaining balanced bisections.

A recursive greedy algorithm is also presented to improve the partition solutions given by the ant colony optimization algorithm and will swap the smallest generated discontinuity in each set of partitions.

A trained feedforward neural network predictor is used to estimate the number of elements which will be generated within each element of the coarse mesh after mesh refinement is carried out. This information is used to partition a coarse mesh using the proposed method based on the estimated number of elements after mesh refinement is conducted (i.e. partitioning is not carried out on the final refined mesh. The optimization method uses the estimated number of elements which will be generated after mesh generation is carried out and partitions the coarse mesh.)

The presented case study demonstrates the efficiency of the proposed method in comparison with a well known mesh decomposing algorithm. The predictive ant colony optimization technique produced good-quality solutions in short period of time.

7. References

Bahreininejad, A. & Topping, BHV. (1996). Finite element mesh partitioning using neural networks. *Advances in Engineering Software*, 27, 103-115

Bahreininejad, A. (2004). A hybrid ant colony optimization approach for finite elements mesh decomposition. *Structural and Multidisciplinary Optimization,* 28, 5, 307-316

Bahreininejad, A. & Hesamfar, P. (2006). Subdomain generation using emergent ant colony optimization. *Computers & Structures*, 84, 1719-1728

Bonabeau, E.; Dorigo, M. & Theraulaz, G. (2000). Inspiration for optimization from social insect behavior. *Nature*, 406, 39–42

Bui, TN. & Strite, LC. (2002). An ant system algorithm for graph bisection. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, USA, Morgan Kaufmann, 43-51

Colorni, A.; Dorigo, M. & Maniezzo, V. (1991). Distributed optimization by ant colonies. *Proceedings of the first European Conference on Artificial Life,* USA, 134–142

Dorigo, M. & Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*. 1, 1, 53–66

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

Kuntz, P. & Snyers, D. (1994). Emergent colonization and graph partitioning. *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 494–500

Kuntz, P.; Layzell, P. & Snyers, D. (1997). A colony of ant-like agents for partitioning in VLSI technology. *In: Proceedings of the Fourth European Conference on Artificial Life*, 417-424

Maniezzo, V. & Carbonaro, A. (2001). Ant colony optimization: an overview. *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, 21–44

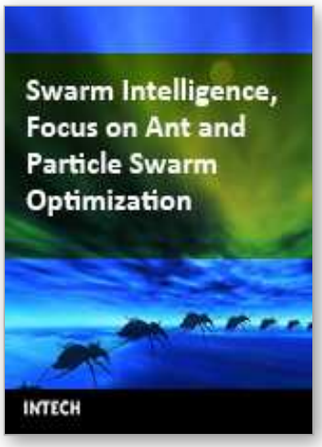
Pao, YH. (1989). Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, USA

Rumelhart, DE.; Hinton, GE. & Williams, RJ. (1986). Learning internal representation by error propagation. *Parallel distributed processing: explorations in the microstructure of cognition*, Rumelhart, DE. & McClelland, JL. (Eds.), MIT Press, USA, 1, 318-362

Simon, H.D. (1991). Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering,* 2, 2-3, 135-138

Topping, BHV. & Bahreininejad, A. (1997). *Neural Computing for Structural Mechanics*, Saxe-Coburg, UK

Topping, BHV. & Khan, AI. (1996). *Parallel Finite Element Computations,* Saxe-Coburg, UK



Swarm Intelligence, Focus on Ant and Particle Swarm Optimization
Edited by FelixT.S.Chan and Manoj KumarTiwari

ISBN 978-3-902613-09-7
Hard cover, 532 pages
Publisher I-Tech Education and Publishing
Published online 01, December, 2007
Published in print edition December, 2007

In the era globalisation the emerging technologies are governing engineering industries to a multifaceted state. The escalating complexity has demanded researchers to find the possible ways of easing the solution of the problems. This has motivated the researchers to grasp ideas from the nature and implant it in the engineering sciences. This way of thinking led to emergence of many biologically inspired algorithms that have proven to be efficient in handling the computationally complex problems with competence such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), etc. Motivated by the capability of the biologically inspired algorithms the present book on "Swarm Intelligence: Focus on Ant and Particle Swarm Optimization" aims to present recent developments and applications concerning optimization with swarm intelligence techniques. The papers selected for this book comprise a cross-section of topics that reflect a variety of perspectives and disciplinary backgrounds. In addition to the introduction of new concepts of swarm intelligence, this book also presented some selected representative case studies covering power plant maintenance scheduling; geotechnical engineering; design and machining tolerances; layout problems; manufacturing process plan; job-shop scheduling; structural design; environmental dispatching problems; wireless communication; water distribution systems; multi-plant supply chain; fault diagnosis of airplane engines; and process scheduling. I believe these 27 chapters presented in this book adequately reflect these topics.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ardeshir Bahreininejad (2007). Finite Element Mesh Decomposition Using Evolving Ant Colony Optimization, Swarm Intelligence, Focus on Ant and Particle Swarm Optimization, FelixT.S.Chan and Manoj KumarTiwari (Ed.), ISBN: 978-3-902613-09-7, InTech, Available from: http://www.intechopen.com/books/swarm_intelligence_focus_on_ant_and_particle_swarm_optimization/finite_element_mesh_decomposition_using_evolving_ant_colony_optimization

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen