International Journal of Pure and Applied Mathematics

Volume 5 No. 2 2003, 141-157

A MULTILEVEL ANT-COLONY OPTIMIZATION ALGORITHM FOR MESH PARTITIONING

Peter Korošec[1], Jurij Šilc[2] §, Borut Robič[3]

[1,2]Computer Systems Department
Jožef Stefan Institute
Jamova 39, SI-1001 Ljubljana, SLOVENIA
[1]e-mail: peter.korosec@ijs.si
[2]e-mail: jurij.silc@ijs.si

[3]Faculty of Computer and Information Science
University of Ljubljana
Tržaška 25, SI-1001 Ljubljana, SLOVENIA
e-mail: borut.robic@fri.uni-lj.si

Abstract: In this paper we present a multilevel ant-colony optimization algorithm, which is a relatively new metaheuristic technique for solving combinatorial optimization problems. We apply this algorithm to the mesh partitioning problem, which is an important problem with extensive applications in various real world engineering problems. The algorithm outperforms the classical k-METIS and Chaco algorithms. In addition, it is comparable even to the combined evolutionary/multilevel scheme used in JOSTLE algorithm.

AMS Subject Classification: 68W20
Key Words: mesh partitioning, randomized algorithm, multilevel algorithm, ant-colony optimization

1. Introduction

A powerful way of solving general problems related to physical models is the finite element method (FEM), which is used for solving partial differential equa-

Received: January 14, 2003

§Correspondence author

tions. A drawback of the FEM is its high computational cost of solving large matrix inversions when the underlying system exhibits no general regularity. One way of improving the speed of such computations is by partitioning the problem and solving it on a parallel computer. For parallelizing FEM, the *domain decomposition* is usually used, where the FEM mesh is split into small submeshes, and each submesh is delivered to a separate processor. This process is known as the domain decomposition. However, the edges shared by neighboring domains require the processes to exchange data. An optimization of the parallel structure of the code to minimize such cross-process data flow is therefore essential for achieving good scalability over a large number of processors.

An important component for mesh partitioning in the domain-decomposition method is the *graph-partitioning* problem. Unfortunately, finding the optimum partition of a mesh under several constraints is known to be an NP-hard problem (discussed later in the paper), which makes it impossible to find optimum solutions for them in time, which is polynomial in the length of the input (unless $P = NP$). As a result, heuristic approaches are normally used to solve mesh-partitioning problems.

Recall that a heuristic technique is one in which "a rule of thumb" is used to speed up the computation of a solution for hard problems. Metaheuristic is "a heuristic for managing heuristics" that controls the activities of one or more heuristics, and usually leads to better solutions than those found with a single heuristic.

Most graph-partitioning methods use either a constructive or an iterative improvement approach. These approaches are very different: the constructive approach works on partial solutions trying to extend them in the best possible way to complete problem solutions, while the iterative improvement approach moves in the search space of complete solutions trying to find the best one [15]. Of course, there are other possibilities, such as the decomposition approach, where we try to distribute the process into smaller tasks that are solved separately.

The *Ant-Colony Optimization* (ACO) uses a metaheuristic approach for solving hard combinatorial optimization problems [3]. The idea for the ACO method came from observing real ants at gathering food and their communication through pheromone trails. ACO is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. ACO implements a randomized construction heuristic that makes probabilistic decisions as a function of artificial pheromone trails, which are determined by the pheromone intensity and heuristic information based on the input data of the problem to be solved, if they are available.

There are two important problem-classes of current ACO applications the static and dynamic combinatorial optimization problems. Static problems are those whose topology and cost do not change while the problems are being solved (e.g., the traveling salesman problem), while in dynamic problems the topology and costs can change while the solutions are being built (e.g., network routing). From a high-level perspective these two classes are very similar, though one can notice many differences in the details.

We are interested in a specific version of the partitioning problem called the *mesh-partitioning problem*, which involves partitioning a mesh into a given number of parts such that each part has roughly the same number of elements, and the number of nodes/edges that straddle the parts is minimized. The reason for doing this is that the program and its associated data must be distributed between processors so that the communication requirements are minimized. One such example is the FEM. The reduction of communication (while balancing workload) in a multiprocessor speeds up the method.

In this paper we investigate the ACO algorithm for mesh partitioning. We suggest modifications to improve this algorithm and evaluate these modification experimentally.

The rest of the paper is organized as follows: In Section 2 we describe the mesh-partitioning problem and introduce the terminology used throughout the paper. Section 3 addresses the ACO metaheuristic with a proposed Multilevel ACO algorithm (MACO) for solving the mesh-partitioning problem. The experimental results are discussed in Section 4, and this is followed by a conclusion.

2. Mesh Partitioning

We focus on the partitioning itself and do not deal with the mesh-generating problem [8, 20], because of its complexity. The mesh-partitioning problem for a non-overlapping partition of an unstructured FEM mesh can be formulated as a graph-partitioning problem.

The FEM mesh is associated with a graph $G(V, E)$ that consists of vertices and edges connecting them, where every vertex v has weight 1 and corresponds to an element of the mesh. An edge between two vertices v and u indicates that the two corresponding elements are neighbors. There are few criteria for determining whether two elements of a mesh are neighbors [2]. The criterion used in this paper is: Two elements of a mesh are considered neighbors if they share a common side in 3D or a common edge in 2D (see Figure 1).

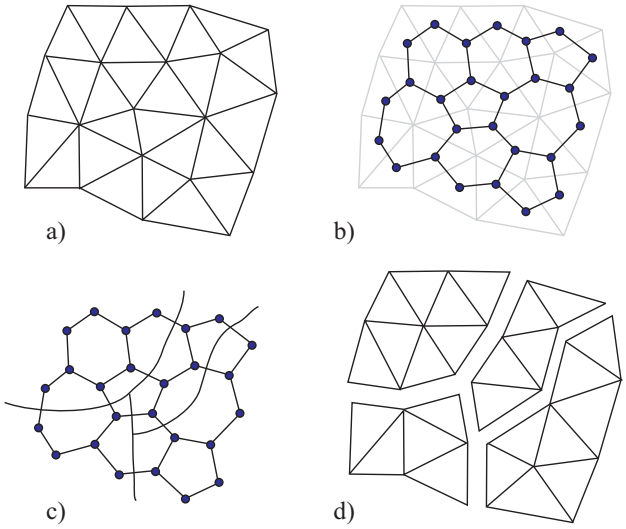


Figure 1: Mesh partitioning: a) mesh; b) mesh with induced graph; c) graph partitioning; d) partitioned into submeshes.

Problem definition. Let $G(V, E)$ be an undirected graph consisting of a non-empty set V of vertices and a set $E \subseteq V \times V$ of edges. A k-partition D of G comprises k mutually disjointed subsets $D_1, D_2, \ldots, D_k$ (called domains) of V whose union is V. The set of edges that connect the different domains of a partition D is called an edge-cut. A partition D is balanced if the sizes of the domains are roughly the same, i.e., if $b(D) = \max_{1 \leq i \leq k} |D_i| - \min_{1 \leq i \leq k} |D_i| \approx 0$. The graph-partitioning problem is to find a balanced partition with a minimum edge-cut, denoted by $\zeta(D)$.

Problem complexity. Assume that the k-partition of $G(V, E)$ is to be found. Let us write $n = |V|$ and suppose that $n = pk$ for some integer p. There are $\binom{n}{p}$ ways to choose the first domain, $\binom{n-p}{p}$ ways to choose the second domain, etc. Hence, there are $\frac{1}{n!}\binom{n}{p}\binom{n-p}{p} \ldots \binom{2p}{p}\binom{p}{p}$ different k-partitions of G. This expression may represent a very large number. For example, in a graph with 80 vertices there can be as many as 10^{61} balanced 8-partitions. Using Stirling's formula it is easy to see that this expression is bounded below by $\Omega(k^n)$. A systematic search for the best k-partition is, therefore, not practical. On the other hand, there is an upper bound on the time complexity of the k-partitioning problem. Indeed, the problem is known to be NP-hard [6]. With this we have shown that mesh partitioning is also NP-hard. Since the space of feasible solutions for the graph-partitioning problem is prohibitively large we

are forced to recourse to heuristic approaches, which reduce or bound the space to be searched.

3. Multilevel Ant-Colony Optimization

The artificial ants used in the ACO are stochastic constructive procedures that build solutions with the help of probability. While building solutions they consider the heuristic information of the problem and the pheromone trails that are being dynamically changed.

```
procedure ACO_Metaheuristic
   while termination condition not satisfied do
      Schedule_Activities
         Manage_Ants_Activity();
         Evaporate_Pheromone();
         Daemon_Actions();
      end Schedule_Activities
   endwhile
end ACO_Metaheuristic.
```

Figure 2: An ACO metaheuristic

The main concept of the ACO metaheuristic (Figure 2) is foraging and gathering food. This involves moving the ant through the graph from the starting vertex to one of the ending vertices. More specifically, ant k in step t moves from vertex i to vertex j with a probability given by:

$$p_{ij,k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{l \in N_{i,k}} \tau_{il}^{\alpha}(t)\eta_{il}^{\beta}}, & \text{where } j \in N_{i,k}\,, \\ \\ 0, & \text{where } j \notin N_{i,k}\,, \end{cases}$$

where η_{ij} is a priori available heuristic information, α and β are two parameters that determine the relative influence of the pheromone trail $\tau_{ij}(t)$ and heuristic information, respectively, and $N_{i,k}$ is the feasible neighborhood of vertex i. Parameters α and β have the following influence on the algorithm's behavior. If $\alpha = 0$ then only heuristic information is considered, on the other hand, if $\beta = 0$ then only pheromone information is at work. Once an ant builds a

solution, or while a solution is being built, the pheromone is being deposited (on nodes or connections) according to the evaluation of a (partial) solution. The solution construction ends when an ant comes to the ending vertex (where food lies).

Besides the main concept of foraging and gathering food, we have also two activities called pheromone-trail evaporation and daemon action (the latter is optional). Pheromone-trail evaporation is a procedure that simulates the reduction of pheromone intensity. This is needed so that we avoid a too rapid convergence of the algorithm towards a sub-optimal solution. Daemon actions can be used to implement centralized actions that cannot be performed by ants.

As we can see from the pseudo code, the `Schedule_Activities` construct does not specify how the three included activities should be scheduled or synchronized. This means it is up to the programmer to specify how these procedures will interact (parallel or independent).

The basic algorithm. The basic idea of the main algorithm is very simple [14]. We have two or more colonies of ants that are competing for food. In our case food are the vertices of the graph.

First we map the graph onto the grid, which represents the ants' habitat (a place where the ants can move). There are many possibilities as to how a graph can be mapped, but, for our example, we will consider a random mapping. Ants are placed into their nest locus from where they start their foraging and gathering of food.

The ants on the grid move in three possible directions (forward, left and right). The decision in which direction an ant will move is defined by the probability of movement. A cumulative probability distribution is used to decide which direction is chosen. When an ant tries to move off the grid, it is forced to move left or right with equal probability. When an ant finds food it tries to pick it up. At first it checks if the quantity of the temporarily gathered food in its nest is not on the limit (the capacity of storage is limited due to the problem constraints). If the limit is not reached, then the weight of the food is calculated from the number of cut edges created by assigning the selected vertex to the partition associated with the nest of the current ant, otherwise the ant moves in a randomly selected direction. If the weight of the food is too heavy for one ant to pick it up (and not too heavy for a few ants to lift it up) then an ant sends a help signal within the radius of a few cells. So if ants are in the neighborhood, they will help this ant to carry the food to the nest locus. On the way back to the nest locus an ant deposits pheromones on the trail that it is making, so the other ants can follow its trail and gather more food from that, or a nearby, cell. When an ant reaches the nest locus it drops the food in

the first possible place around the nest (in a clockwise direction). After an ant drops its food it starts a new round of foraging. Of course ants can also gather food from other nests. When an ant tries to pick up a food from the other nests it a performs the same procedure as if it was gathering for food, except when the food is to heavy to pick it up, it does not send a help signal but moves on. With this, we significantly improve our temporary solution.

```
procedure ACO
 Initialize()
 while ending condition not satisfied do
  for all ants do
   if carrying food then
    if in nest locus then Drop_Food()
    else Move_to_Nest()
   else if food here then Pick_Up_Food()
    else if food ahead then Move_Forward()
     else if in nest locus then Move_To_Away_Pheromone()
      else if help signal then Move_To_Help()
       else Follow_Strongest_Forward_Pheromone()
  endfor
  for all grid cells do Evaporate_Pheromone() endfor
 endwhile
end ACO.
```

Figure 3: The basic ACO algorithm

As we have already mentioned, there are some constraints that are imposed on our algorithm. First, we must mention the colony's storage-capacity constraint. This is implemented so no single colony can gather all the food into its nest and the appropriate balance between domains is maintained. When the pheromone intensity of a certain cell drops below a fixed value, then that cell's pheromone intensity is restored to the initial value. With this we maintain a high exploration level. There can be only a limited number of vertices put in a single cell. Each ant can carry only a limited number of pieces of food. Here we would like to mention one more feature. While the ant is bringing the food towards the nest, this food is in some kind of tabu state (i.e., the food is not accessible to the other ants). This is very helpful when the algorithm tries to climb out of the local minimum. To even enforce this we added a short tabu list, which consists of last m food pieces that we moved.

We have made several experiments to see how successful the ACO algorithm is at solving graph-partitioning problem [12, 13]. But we have noticed that algorithm performs well only on smaller graphs ($n < 500$). Therefore, we have enhanced our basic ACO algorithm with a multilevel technique [18, 19].

The multilevel paradigm. One already established effective way to speed up and globally improve partitioning method is the use of multilevel techniques. The basic idea is to group vertices together to form clusters. These clusters define a new graph. We recursively iterate this procedure until the graph size falls below some threshold. This is followed by successive refinement of these coarsen graphs. This procedure is known as the multilevel paradigm. The multilevel idea was first proposed by Barnard and Simon [1], as a method of speeding up spectral bisection and improved by Hendrickson and Leland [7] who generalized it to encompass local refinement algorithms.

Implementation. The implementation itself consists of two parts: graph contraction and partition expansion. In first part, we create a coarser graph $G_{\ell+1}(V_{\ell+1}, E_{\ell+1})$ from $G_\ell(V_\ell, E_\ell)$ by finding the maximal independent subset of graph edges and then collapse them. Each selected edge is collapsed and the vertices $u_1, u_2 \in V_\ell$ that are at either end of it are merged into new vertex $u \in V_{\ell+1}$ with weight $|v| = |u_1| + |u_2|$. Edges that have not been collapsed are inherited by the new graph $G_{\ell+1}$ and the edges that become duplicated are merged and their weight summed. Because of inheritance the total weight of the graph remains the same and the total edge weight is reduced by an amount equal to the weight of the collapsed edges, which have none impact on graph balance or edge cut. In second part, we expand already optimized partition (with the ACO algorithm) of graph G_ℓ. The optimized partition must be interpolated onto its parent graph $G_{\ell-1}$. Because of the simplicity of coarsening in first part, the interpolation itself is very trivial. So, if vertex $v \in V_\ell$ belongs to domain D_i then after refinement the matched pair $u_1, u_2 \in V_{\ell-1}$ that represents v, will also be in D_i. We are expanding the graph to its original size. On every level ℓ of our expansion we run our basic ACO algorithm. We call this approach as *Multilevel Ant-Colony optimization* (MACO).

Due to large graphs and increased number of levels, the number of vertices in a single cell increases rapidly. Therefore, we introduced a method (called *bucket sort*) that accelerates and improves algorithm's convergence by choosing the most "promising" vertex from the cell.

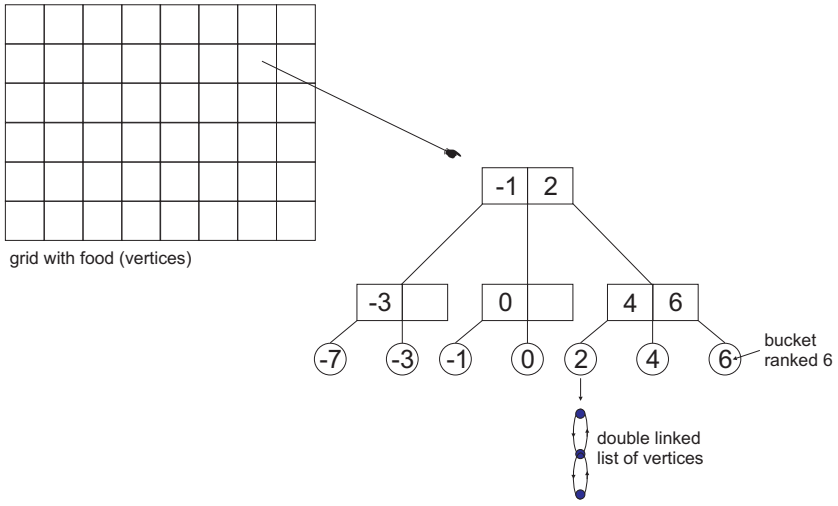


Figure 4: Bucket sorting

Bucket sorting. The bucket sort has become an essential tool for efficient and rapid sorting and adjustment of vertices by their gain. The main idea was first introduced by Fiduccia and Mattheyses [5]. The basic idea is that all vertices of given gain g are put together in a "bucket" ranked g. Problem of finding a vertex with maximum gain converts into finding the non-empty bucket with the highest rank and picking a vertex from it. If a chosen vertex migrates from one domain to another then only its gain and the gains of all its neighbors have to be recalculated and put back into appropriate buckets.

In our implementation each bucket is represented by a double linked list of vertices. Because of the multilevel process, it often happens that the potential gain values are dispersed on a wide range. For this reason we have introduced 2-3 tree. With this we have rid of large and sparse arrays of pointers. We store the non-empty buckets into the 2-3 tree, so each leaf in the tree represents a bucket. For even faster searching, we have made one 2-3 tree for each colony on every cell that have vertices on it (see Figure 4). With this we have fasten the search, add and delete operations.

Abbreviation	Algorithm	Reference
Ch2.0	Chaco v. 2.0 - multilevel Kernighan-Lin (recursive bisection)	[7]
JE	JOSTLE Evolutionary - combined evolutionary/multilevel scheme	[16]
Grdy	Greedy algorithm - (as implemented within JOSTLE)	[4]
iJ	Iterated JOSTLE - iterated multilevel Kernighan-Lin (k-way)	[17]
MRSB	Multilevel Recursive Spectral Bisection	[1]
pM4.0	p-METIS v. 4.0 - multilevel Kernighan-Lin (recursive bisection)	[9]

Tbale 1: Mesh-partitioning algorithms

4. Performance Evaluation

Let us now present and discuss the results of the experimental evaluation of our MACO algorithm in comparison to the well-known partitioning programs *k-METIS 4.0* [10], *Chaco 2.0* [7], and results obtained from Graph Partitioning Archive at the end of the year 2002 (Table 1).

Benchmark graphs used in our experiment were acquired from the above URL address and are described in Table 2.

We partitioned each of the graphs into two and four domains ($k = 2, k = 4$). Each score is described with an edge cut $\zeta(D)$ and a balance $b(D)$. Balance is defined as the difference (in the number of vertices) between the largest and the smallest domain. We decided to restrict balance with $b(D) \leq 0.002\frac{|V|}{k}$. The results of our experiment are shown in Table 4.

Graph $G(V, E)$	$\|V\|$	$\|E\|$
uk	4,824	6,837
add20	2,395	7,462
add32	4,960	9,462
3elt	4,720	13,722
data	2,851	15,093
whitaker3	9,800	28,989
crack	10,240	30,380
fe_4elt2	11,143	32,818
cs4	22,499	43,858
4elt	15,606	45,878
cti	16,840	48,232
fe_sphere	16,386	49,152
wing_nodal	10,937	75,488
bcsstk33	8,738	291,583

Table 2: Benchmark graphs

It is clear that the MACO algorithm performed very well. One can notice that our MACO algorithm is superior to classical *k-METIS* [10] and *Chaco* [7] algorithms (Table 4). Furthermore, it is even comparable to combined evolutionary/multilevel scheme used in JOSTLE Evolutionary algorithm [16]. MACO algorithm returned also some solutions that are better then currently available solutions in the Graph Partitioning Archive (Table 3).

The experimental results have been obtained with the implementation of the MACO algorithm that also includes a visualization tool to ease the user in selecting the appropriate parameters of the algorithm (see Figure 5).

Graph	k	Algorithm	$\zeta(D)$	$b(D)$
add20	2	Ch2.0	630	1
add20	4	JE	1,203	1
data	2	Grdy	209	1
data	4	iJ	432	1
3elt	2	JE	**90**	0
3elt	4	JE	**201**	0
uk	2	JE	22	0
uk	4	JE	**44**	0
add32	2	Ch2.0	11	0
add32	4	pM4.0	**37**	0
bcsstk33	2	Ch2.0	**10,199**	0
bcsstk33	4	iJ	**22,115**	1
whitaker3	2	JE	127	0
whitaker3	4	JE	**382**	0
crack	2	JE	**184**	0
crack	4	JE	**368**	0
wing_nodal	2	JE	**1,707**	1
wing_nodal	4	JE	**3,581**	1
fe_4elt2	2	MRSB	**130**	1
fe_4elt2	4	JE	**349**	1
4elt	2	JE	**139**	0
4elt	4	JE	**327**	1
fe_sphere	2	JE	**386**	0
fe_sphere	4	N.A.	**773**	1
cti	2	JE	334	0
cti	4	JE	**977**	0
cs4	2	JE	**372**	1
cs4	4	JE	**964**	1

* `http://staffweb.cms.gre.ac.uk/~wc06/partition`

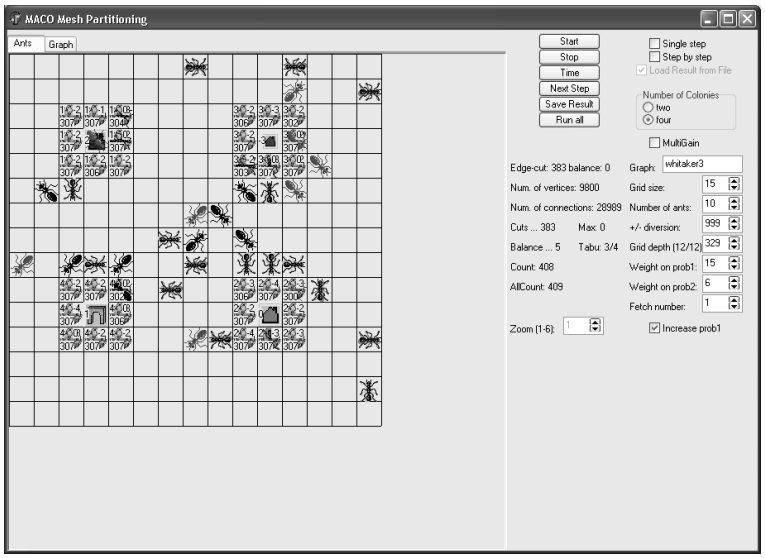
Table 3: Results from Graph Partitioning Archive*

		k-METIS 4.0 [10]		Chaco 2.0 [7]		MACO	
Graph	k	$\zeta(D)$	$b(D)$	$\zeta(D)$	$b(D)$	$\zeta(D)$	$b(D)$
add20	2	774	23	630	1	**601**	5
add20	4	1,214	45	1,242	1	**1,196**	3
data	2	253	67	210	1	**199**	1
data	4	486	42	444	1	**424**	1
3elt	2	148	72	124	0	**90**	0
3elt	4	250	63	258	0	225	2
uk	2	33	126	23	0	**20**	2
uk	4	50	47	60	0	50	2
add32	2	12	62	11	0	**10**	2
add32	4	38	59	53	0	40	5
bcsstk33	2	13,393	139	**10,199**	0	10,222	12
bcsstk33	4	22,909	129	25,529	1	22,632	14
whitaker3	2	135	96	135	0	**126**	16
whitaker3	4	407	97	439	0	386	6
crack	2	221	248	209	0	**184**	0
crack	4	478	106	457	0	381	8
wing_nodal	2	1,892	315	1,747	1	1,711	23
wing_nodal	4	3,898	163	3,817	1	3,619	11
fe_4elt2	2	132	297	**130**	1	**130**	1
fe_4elt2	4	400	87	378	1	350	2
4elt	2	149	104	242	0	**139**	8
4elt	4	385	197	416	1	389	13
fe_sphere	2	444	418	422	0	402	4
fe_sphere	4	828	46	835	1	810	15
cti	2	401	300	369	0	**332**	24
cti	4	1,104	156	1,000	0	1,045	14
cs4	2	397	653	418	1	397	1
cs4	4	1,102	312	1,135	1	1,038	21

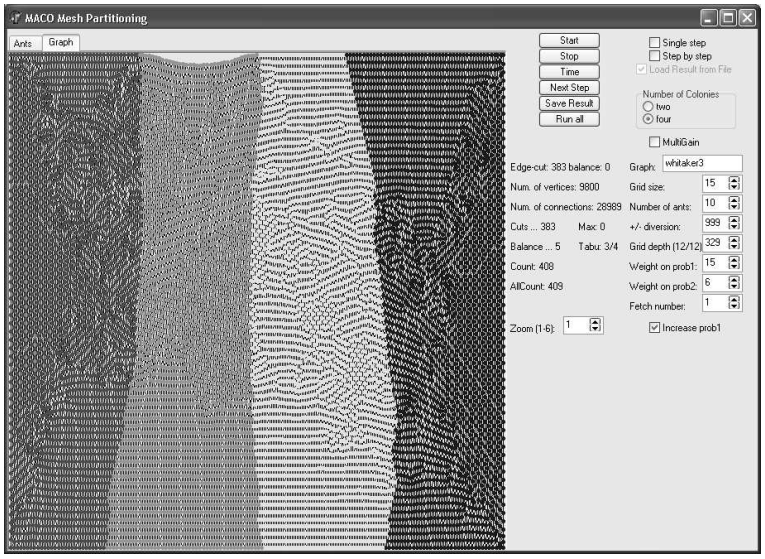
Table 4: Experimental results

5. Conclusions

The graph-partitioning problem is an important component for mesh partitioning in the domain-decomposition method. The Ant-Colony Optimization method uses a metaheuristic approach for solving hard combinatorial optimization problems. The goal of this paper was to investigate an Ant-Colony Optimization algorithm for mesh partitioning, to suggest modifications to improve this algorithm, and to evaluate them experimentally. Experimental results show that the basic Ant-Colony Optimization algorithm performed very well on small



a)



b)

Figure 5: MACO mesh-partitioning visualization: a) ant colonies; b) corresponding submeshes.

or medium-sized graphs ($n < 500$). With larger graphs, which are often encountered in mesh partitioning, we had to use a multilevel method to produce results that were competitive with the results given by other algorithms. The Multilevel Ant-Colony Optimization method is a very promising method that needs to be thoroughly investigated.

There are many possibilities for improving our algorithm. One possibility is in the mapping of the graph onto the grid: with a proper mapping convergence the results can be improved. The use of load balancing method between levels would be also very promising way to go. The next possibility is in determining which and how many vertices from the cell will be picked and with what probability. Here, the Kernighan-Lin gain method [11] might be used. We could also add some daemon actions, like the min-cut algorithm, to improve solutions during crossing from one level to another. And, finally, we could change the way the pheromone is evaporated, deposited and restored.

There is a wide range of possibilities to be considered in the future. One of the most appealing is a merger of the MACO method with some other method through daemon actions and parallel implementation of MACO algorithm.

References

[1] S.T. Barnard, H.D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, *Concurrency: Practice & Experience*, **6** (1994), 101–117.

[2] N. Bouhmala, X. Cai, Partition of unstructured finite element meshes by a multilevel approach, *Lect. Notes Comput. Sc.*, **1947** (2001), 187–195.

[3] M. Dorigo, T. Stützle, The ant colony optimization metaheuristic: algorithms, applications, and advances, Tech. Rep. IRIDIA-2000-32 (Université Libre de Bruxelles, 2000).

[4] C. Farhat, A simple and efficient automatic FEM domain decomposer, *Comput. & Structures*, **28** (1988), 579–602.

[5] C.M. Fiduccia, R.M. Mattheyses, A linear time heuristic for improving network partitions, In: *Proceedings of the 19-th IEEE Design Automation Conference* (1982), 175–181.

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

[7] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, In: *Proceedings of the Supercomputing'95* (1995).

[8] K. Ho-Le, Finite element mesh generation methods: a review and classification, *Comput. Aided Design*, **20** (1988), 27–38.

[9] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.*, **20** (1998), 359–392.

[10] G. Karypis, V. Kumar, Multilevel k-way partioning scheme for irregular graphs, *J. Parallel Dist. Comput.*, **48** (1998), 96–129.

[11] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graph, *Bell Sys. Tech. J.*, **49** (1970), 291–307.

[12] P. Korošec, J. Šilc, B. Robič, An Ant-Colony optimization approach to the mesh-partitioning problem, In: *Parallel Numerics '02* (Ed-s: R. Trobec, P. Zinterhof, M. Vajteršic, A. Uhl), University of Salzburg and Jozef Stefan Institute (2002), 123–132.

[13] P. Korošec, J. Šilc, B. Robič, Mesh partitioning: a multilevel Ant-Colony optimization algorithm, In: *Proceedings of the 6-th International Workshop on Nature Inspired Distributed Computing* (April 2003).

[14] A.E. Langham, P.W. Grant, Using competing ant colonies to solve k-way partitioning problems with foraging and raiding strategies, *Lect. Notes Comput. Sc.*, **1674** (1999), 621–625.

[15] J. Šilc, P. Korošec, B. Robič, An experimental evaluation of modified algorithms for the graph partitioning problem, In: *Proceedings of the 17-th International Symposium on Computer and Information Sciences* (October 2002), 120–124.

[16] A.J. Soper, C. Walshaw, M. Cross, A combined evolutionary search and multilevel optimisation approach to graph partitioning, *Tech. Rep.*, **00/IM/58**, Comp. Math. Sci., Univ. Greenwich, London (2000.

[17] C. Walshaw, Multilevel refinement for combinatorial optimisation problems, *Tech. Rep.*, **01/IM/73**, Comp. Math. Sci., Univ. Greenwich, London (2001).

[18] C. Walshaw, M. Cross, Mesh partitioning: a multilevel balancing and refinement algorithm, *Tech. Rep.*, **98/IM/35**, Centre for Numerical Modelling and Process Analysis, University of Greenwich (1998).

[19] C. Warshaw, M. Cross, Mesh partitioning: a multilevel balancing and refinement algorithm, *SIAM J. Sci. Comput.*, **22** (2001), 63–80.

[20] J. Żola, R. Wyrzykowski, J. Šilc, G. Papa, Genetic algorithms as a method for finite element mesh smoothing, In: *Proceedings of the 3-rd International Conference on Parallel Processing and Applied Mathematics* (September 1999), 637–644.