BULETINUL STIINTIFIC al Universitatii "Politehnica" din Timisoara, ROMANIA,
Seria AUTOMATICA si CALCULATOARE
SCIENTIFIC BULLETIN of "Politehnica" University of Timisoara, ROMANIA,
Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. xx (yy), Fasc. z, 20vv, ISSN 1224-600X

Nature Inspired Metaheuristics for Task Scheduling in Static and Dynamic Computing Environments

Flavia Zamfirache*, Daniela Zaharie* and Ciprian Craciun*

* Department of Computer Science, West University of Timisoara, Faculty of Mathematics and Informatics, Blvd. V. Parvan 4
Timisoara 300223, Romania, Phone: (0)256-592155, Fax: (0)256-592316, E-Mail: {zflavia,dzaharie,ccraciun}@info.uvt.ro

Abstract **–** ***Background:*** ***Finding optimal assignments of tasks to processing elements in heterogeneous computing environments is a challenging optimization problem which attracted a lot of researchers during the last decades. The aim of this paper is to analyze the behavior of two nature inspired approaches for task scheduling in static and in dynamic environments.***
Methods: ***An evolutionary and an ant based schedulers are proposed. For each one the selection of adequate operators is discussed and some memory based mechanisms are investigated in order to asses their ability to deal with the dynamic nature of the environment. The numerical analysis is based on a well-known benchmark for task scheduling in heterogeneous environments and the dynamic character of the environment is simulated randomly marking some resources as unavailable.***
Results: ***Results obtained by three types of experiments are provided. The first experiment aims to assess the effectiveness of the schedulers in the case of static computing environments. The second experiment is focused on the comparison between the static and dynamic variants of the schedulers tested in the case of the simulated dynamic environment. The last experiment provides results of a robustness analysis.***
Conclusions: ***The numerical results illustrate that the nature inspired schedulers produce good and robust schedules especially in the case of heterogeneous computing environments characterized by inconsistency. In the case of dynamic environments the memory techniques introduced in the nature inspired schedulers proved to be beneficial as long as the ratio of processors which become unaivalable between successive scheduling events is around 10%.***

Keywords: ***task scheduling, heterogeneous computing environments, evolutionary algorithms, ant colony optimization, dynamic optimization***

I. INTRODUCTION

The task scheduling problem, i.e. the assignment of tasks to resources such that some quality of services criteria are optimized, attracted a lot of attention lately [1][2]. As a consequence, currently there are a lot of scheduling algorithms addressing different variants of the problem. The task scheduling variants differ with respect to the tasks properties, to the computing environment characteristics and to the scheduling process particularities. Thus the set of tasks to be scheduled (sometimes called application job or meta-task) could consist of independent tasks or of inter-related ones. In the first case the scheduling problem belongs to the unconstrained optimization class, while in the second case it is a constrained optimization problem. The computing environment can consist of homogenous computing nodes (as in the cases of computational clusters) or of heterogeneous resources (as in the case of grid or cloud computing). Moreover the availability of resources can vary in time, i.e. some of them can become unavailable. In this case the computing environment is a dynamic one with elements appearing and disappearing from the resource pool making the optimization problem a dynamic one. On the other hand there are are different approaches in mapping tasks to resources [3]: online mapping, also called dynamic scheduling, when the tasks are mapped as they arrive; batch mapping, when the scheduler waits until a given number of tasks arrived; pseudo-batch mapping characterized by the fact that always when an event occurs (e.g. after a given amount of time) all unscheduled tasks are (re)scheduled. In the case of batch mapping the number of tasks to be scheduled is constant while in the case pf pseudo-batch mapping the number of tasks is variable.

This paper deals with the batch mapping of independent tasks in a heterogeneous computing environment characterized by a variable set of available resources, i.e. batch mapping in a dynamic environment. For instance, let us consider a meta-task consisting of a set of independent tasks which should be periodically executed on a distributed computing environment. This means that there are some scheduling events triggered each time the meta-task should be scheduled again. Between two consecutive scheduling events the status of the computing environment can change, meaning that some machines become unavailable while other ones become available. Moreover, the computing environment is usually heterogeneous meaning that the execution time of a given task is different on different machines. Thus a good schedule corresponding

to a given configuration is not necessarily good for another configuration. There are two main approaches to deal with such a problem: (i) at each scheduling stage a new schedule is started from scratch; (ii) in order to construct the schedule corresponding to a new stage, some information from the previously generated schedule are used.

The main questions arising in the second case are: (i) what information from the previous scheduling stages should be used in order to construct a new schedule?; (ii) what mechanisms should be involved in the metaheuristic used to construct the schedule in order to make it appropriate for dynamic environments?; (iii) what is the threshold corresponding to the percent of changes in the list of available machines which allows to obtain benefits from using information from the previously obtained schedules?

In this paper we extend the analysis initiated in [4] concerning the behavior of several mechanisms of exploiting information from previous scheduling stages applied to two nature inspired schedulers: an evolutionary one and an ant colony optimization one. The experimental analysis is conducted on the benchmark data provided in [3] and the dynamic character of the computing environment is simulated by randomly marking some resources as unavailable.

The rest of the paper is structured as follows. The scheduling problem is described in the next section and Section III presents a short review of existing nature inspired metaheuristics for scheduling. The evolutionary scheduler investigated in this paper is described in Section IV and the ant based scheduler is presented in Section V. The experimental results and their discussion are presented in Section VI and the last section concludes the paper.

II. THE TASK SCHEDULING PROBLEM

The task scheduling problem aims to find an allocation of resources to tasks such that the tasks requirements (hardware, software) and their precedence constraints are satisfied and some quality of service criteria (concerning response/waiting/completion time, load balancing, data transfer costs etc) are optimized. In this paper we shall consider that the tasks are independent and the criteria to be optimized is the maximal execution time over all resources, i.e. *makespan*. The assignment of tasks is based on knowing some estimation of the execution times of the different tasks on different resources.

To be more specific, let us consider a set of n tasks, $\{t_1,\ldots,t_n\}$ to be scheduled on a set of $m < n$ processors (machines), $\{p_1,\ldots,p_m\}$. Let us suppose that for each pair (t_i, p_j) is known an estimation $ET(i, j)$ of the time needed to execute the task t_i on the processor p_j.

A schedule is an ordered sequence of n pairs associating processors to tasks, $S = ((t_{i_1}, p_{j_1}),\ldots,(t_{i_n}, p_{j_n}))$. The tuple $(i_1,\ldots,i_n)$ denotes a permutation of order n specifying the order of allocation of the tasks. In the case of independent tasks this order does not influence the quality of the schedule but it is important in the case of dependent tasks when some order constraints should be satisfied.

There are different quality measures used to evaluate the the schedule [6]: makespan, flowtime, imbalance, tardiness, lateness. By far the most used measure is the makespan, i.e. the longest execution time over the entire set of processors. The makespan can be defined as follows:

$$makespan(S) = \max_{k=1,n}(CT(i_k, j_k)) \quad (1)$$

where $CT(i_k, j_k)$ denotes the completion time of task i_k on the processor j_k and can be defined by:

$$CT(i_k, j_k) = T(j_k, S, k-1) + ET(i_k, j_k) \quad (2)$$

with $T(j_k, S, k-1)$ denoting the time moment since the processor j_k is free under the assumption that the tasks $t_{i_1},\ldots,t_{i_{k-1}}$ are already executed. This is related to the completion time of the task scheduled at a previous step on processor j_k, i.e.

$$T(j,S,k) = \begin{cases} T(j,S,k-1) + ET(i_k, j) & if\ j = j_k\ in\ S \\ T(j,S,K-1) & otherwise \end{cases} \quad (3)$$

At the beginning of the scheduling process all processors are considered to be free, i.e. $T(j,S,0) = 0$ for any j. The aim of the scheduling process analyzed here is to identify the schedule S^* having the property that $makespan(S^*) = \min_S makespan(S)$.

III. SHORT REVIEW OF NATURE-INSPIRED METAHEURISTICS FOR TASK SCHEDULING

Finding the globally optimal schedule is a difficult problem, therefore near-optimal solutions are acceptable in practice. A lot of heuristics that construct such near-optimal solutions have been proposed up to now. A comparison of 11 such heuristics is presented in [3] for several sets of simulated execution times generated such they express different characteristics of the computing environment and of the set of tasks. The tested heuristics include Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Min-Min, a genetic algorithm, simulated annealing and tabu-search. The heuristics with a consistently good behavior were Min-Min and the genetic algorithm. The Min-Min heuristics, which we also used for comparisons, is

a fast greedy approach based on the idea of finding the minimum completion time for each task and then assigning the task with the smallest completion time to the processor ensuring this time. Since the results on effectiveness of genetic algorithms in tasks scheduling were presented in [3] a lot of other nature inspired metaheuristics were successfully applied in solving different variants of the scheduling problem: genetic algorithms [7], [8], [9], [10], ant colony optimization [11], [12], differential evolution [13], particle swarm optimization [14]. Recent reviews of different meta-heuristics for tasks scheduling, including the nature-inspired ones can be found in [1], [2], [15].

Some of the existing approaches deal with the static scheduling problem where the tasks and machines characteristics are known apriori and do not change during the scheduling process. Such approaches are those presented in [3], [15], [12]. On the other hand there are also results on online scheduling, meaning that the tasks arrive sequentially and they are scheduled as they arrive (e.g. [15], [14], [8]). The main approach in online scheduling is that based on managing a queue of tasks for each processor and in ensuring the load balancing by moving tasks from one queue to another one.

One of the main issue addressed in the works studying the applicability of evolutionary algorithms in tasks scheduling is to identify the appropriate operators. An extensive study of different operators is presented in [7] and one of the main conclusion of this study is that the best results are obtained by using a mutation operator that does a rebalancing of the current solution. The aim of rebalancing is to move tasks between processors in order to reduce the makespan. However there is no detailed description of the operation in [7]. The idea of using rebalancing is also exploited in [8]. Therefore we also used the idea of rebalancing the solutions in the Evolutionary Scheduler proposed in Section IV. In [9] is presented a dynamically genetic scheduler that operates in batch mode into an environment with dynamically changing resources that demonstrates the robustness of the evolutionary approaches. In [10] it is proposed a decentralized solution for task scheduling that provides better results for processors load balancing and makespan.

Currently there are several ant based approaches both for static [12], [16], [11] and dynamic scheduling [17]. The common element of all these approaches is the fact the scheduling problem is formulated as a problem of finding a path in a graph. When the tasks are independent this graph is fully connected. A node in the graph can correspond to a pair (task, processor) as in [11], [16] or to a task as in [12]. In the first case the graph has $m \cdot n$ nodes but a schedule will be a path with n nodes while in the second case a schedule correspond to a path which visits all nodes in the graph. In both cases each ant will build a schedule by visiting exactly n nodes. Each ant passes from one node to another one by using a transition probability which depends on two main factors, one which is based on a local heuristic value related to the current schedule and one related to a somewhat global value combining information from all ants (the so-called pheromone level). The best results reported in the literature [12] were obtained for the ant-based schedulers which are hybridized with some local search operators, which are similar to rebalancing operators used in evolutionary approaches.

IV. AN EVOLUTIONARY SCHEDULER

In this section is presented the evolutionary scheduler proposed in [4] and its adaptation for the case of periodical scheduling in a dynamic environment. The general structure of the Evolutionary Scheduler in the case of static environments is described in Algorithm 1 and the details concerning the solution representation, operators and adaptation to dynamic environments are presented in the following subsections.

Algorithm 1 The general structure of the Evolutionary Scheduler (ES) in static environments

1: Initialize the population
$X(0) \leftarrow \{x_1(0), \ldots, x_N(0)\}$
2: $g \leftarrow 0$
3: Evaluate $X(0)$
4: **while** the stopping condition is false **do**
5: Copy $X(g)$ to $X(g+1)$
6: **for** $i = \overline{1, N}$ **do**
7: Construct the mutant y_i from $x_i(g)$
8: **if** $makespan(y_j) < makespan(x_i(g))$ **then**
9: Replace $x_i(g+1)$ with y_i
10: **Else**
11: **if** $makespan(y_j) = makespan(x_i(g))$ **then**
12: Append y_i to $X(g+1)$
13: **Else**
14: **if** $rand(0,1) < p_r$ **then**
15: Replace $x_i(g+1)$ with a random perturbation of y_i
15: **end if**
17: **end if**
18: **end if**
19: **end for**
20: Select N elements from $X(g+1)$
21: $g \leftarrow g+1$
22: **end while**

A. Representation of schedules

Since a schedule is a set, $S = ((t_{i_1}, p_{j_1}), \ldots, (t_{i_n}, p_{j_n}))$, of pairs (task, processor) each element in the population

consists of a *n*-order permutation $(i_1,\ldots,i_n)$ specifying the tasks execution order and a mapping of tasks to processors, $(j_1,\ldots,j_n)$, where j_k belongs to $\{1,\ldots,m\}$ and denotes the index of the processor on which the task of index i_k will be executed. In the case of scheduling independent tasks it is enough to use just the mapping of tasks to processors but in the general case when the tasks are interrelated, the permutation should also be used in the representation. The evolutionary scheduler was designed to be easily adapted for the general case of interrelated tasks, even if the results presented in this paper are for the case of independent tasks.

B. Population initialization

The initial population consists mainly of randomly generated schedules. A random schedule consists of a random permutation and a random assignment of tasks to processors, both being generated by using uniform distributions. Besides the fully random schedules there are also schedules which are partially generated by random assignment of tasks to processors and partially by using a greedy like assignment based on the estimated completion time of a task on a given processor. More specifically, these schedules are generated by executing the following steps:

- a random index $r \in \{1,\ldots,n\}$ is selected;
- the pairs $(i_1, j_1),\ldots,(i_r, j_r)$ are randomly generated;
- for all remaining $n-r$ tasks, the processor is selected by the criteria of the minimal estimated completion time, e.g. for a task i is selected the processor j satisfying that $ET(i,j) = \min_{l\in M} ET(i,l)$, where M denotes the list of available machines (processors).

In our implementation, 75% of the population elements were randomly initialized while 25% of elements were initialized using the above steps. The idea of this initialization procedure started from using partial components of the schedule generated by the MinMin heuristics. In fact the MinMin schedule has been also introduced in the initial population.

The incorporation of the MinMin schedule (or other schedules obtained by a non-evolutionary heuristic) as a seed in the initial population is a common approach in evolutionary scheduling [7]. The proposed initialization proved to behave significantly better than that based on only random schedules.

C. Mutation operator

Several mutation operators were investigated but finally two of them were selected: a "shift"-based mutation and a "rebalancing"-based mutation. The "shift"-based mutation consists of the following steps:

- extract a random task from the list of assigned tasks and update accordingly the completion time of tasks on the processor corresponding to the selected task;
- assign the selected task to the processor ensuring the minimal completion time;
- append the extracted task to the list of assigned tasks.

If the population element to be mutated consists of the tasks permutation $(i_1,\ldots,i_n)$ and of processors mapping $(j_1,\ldots,j_n)$ and the selected task has the index i_k then by "shift"-based mutation the tasks permutation will become $(i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_n,i_k)$ and the mapping will be $(j_1,\ldots,j_{k-1},j_{k+1},\ldots,j_n,j_k')$ where j_k' is the processor on which the task i_k is reassigned. On the other hand, the "rebalancing"-based mutation does not change the tasks permutation but only the assignment of tasks to processors. The task to be reassigned and the new processor are selected as follows:

- find the processor with the largest completion time;
- select a random task from this processor;
- assign the task to the processor which ensures the largest decrease of the makespan; if there is no processor ensuring a decrease of the current makespan then the mutation is not applied.

Each of these two mutation strategies is applied with a given probability: p_R for "rebalancing" mutation and $1-p_R$ for the "shift" mutation. Remarks on the appropriate values of p_R depending on the particularities of the problem are presented in Section VI. For each element of the population the mutation operator is applied for several times, depending on the mutation probability, p_m.

D. Selection of survivors

After generating the population of mutant individuals the elements of the new generation are selected by comparing each element in the current population with the corresponding mutant. Depending on the quality of these two elements there are three main cases and in each one a specific decision is taken:

- *The mutant is better than the parent*. In this case the mutant unconditionally replaces its parent.
- *The mutant and the parent have the same quality*. The mutant is appended to the population.
- *The mutant is worse than the parent*. An extra mutation is applied, with a probability p_r, to the mutant by reassigning a randomly selected task to a randomly selected processor and it replaces its parent.

The random perturbation specified in the last case has also the aim to stimulate the population diversity. At the end of

the selection process it is possible to have more than *N* elements in the population. To reduce the population size to the initial value some elements should be eliminated. This is usually done by a standard truncation selection.

E. Adaptation to the changes in the environment

In the case of a dynamic environment the scheduling process consists of consecutive scheduling events. At each scheduling event there is a list of available processors which is partially different from the list corresponding to the previous step. If the difference between the list of available processors is not too large then one can exploit the schedules constructed at the previous scheduling event. The main idea of using this information is to keep a percent of the best elements from the previous population and to reinitialize the other elements. The main issue arising in such an approach is that the elements from the previous scheduling event should be adjusted to the current set of available processors. The adjustment is made by assigning each task which was previously assigned to a processor which is no more available to the fittest available processor, i.e. the processor which ensures the minimal completion time for that task. This reassignment of tasks is made sequentially in the order given by the tasks permutation corresponding to the element.

In this paper we analyzed the behavior of two variants of using information from the evolutionary process corresponding to previous scheduling events:

- *Use of good schedules obtained in the previous scheduling event*. In this case a percent of the best solutions obtained in the previous scheduling event are kept and adjusted while all other elements are initialized by applying the strategy used to construct the initial population (e.g. top 10% of the elements are conserved and the other ones are reinitialized).
- *Use of an archive of good schedules obtained in previous scheduling events*. The best schedule constructed at each previous scheduling event is kept in an archive and when a new scheduling event is triggered the population is constructed by taking the elements in the archive and by filling in the rest of the population with newly created elements. The archive has a limited size, thus the old elements are removed once there is no free space to add elements from the current scheduling events.

The behavior of both variants has been analyzed and the main remarks are presented in Section VI. The general structure of the scheduler corresponding to dynamic environments is presented in Algorithm 2.

V. AN ANT BASED SCHEDULER

In this section we present a scheduler based on the ant systems paradigm.

Algorithm 2: The Evolutionary Scheduler in dynamic environments

```
for each scheduling event se do
  Construct the list M of available processors
  if se = 1 then
    Construct the initial population as in Algorithm 1
  Else
    Construct the initial population using information
    from the previous scheduling event (se − 1)
  end if
  Apply the evolutionary steps 4-22 from Algorithm 1
  Update the archive by adding the best schedule
end for
```

This variant of the scheduler corresponding to the case of static environments (described in Algorithm 3) is mainly based on the ideas presented in [12]. The details concerning the pheromone matrix initialization, the construction of schedules and local search step are presented in the following subsections.

Algorithm 3 The general structure of the Ant Scheduler (AS) for the static environment

```
Initialize the pheromone matrix
while the stopping condition is false
  Reset all ants
  for each ant do
    Build a schedule
  end for
  Find the best schedule from the current step (S*)
  Apply rebalancing to the best schedule (S*)
  Update the pheromone matrix using Eq. 6
end while
```

A. Pheromone initialization

The pheromone matrix is initialized with a constant value τ_0. In the experiments, τ_0 was set with value 0.01 following the suggestions in [12]. In order to help the construction of good schedules we used the idea of incorporating information corresponding to a schedule generated by the MinMin heuristic. This has been done by reinforcing the elements in the pheromone matrix which correspond to the MinMin schedule. This is similar with the incorporation of the MinMin schedule in the initial population corresponding to the evolutionary scheduler. The idea to seed some information obtained by a greedy heuristics is a common approach in evolutionary scheduling [7] and it also improves the behavior of the ant based scheduler.

B. Construction of a schedule

Each ant constructs a schedule in n successive steps. At each step, k, each ant makes two choices. First it chooses for each unscheduled task, $i \in U_k$, the processor which

would lead to the smallest completion time, p_{best}^{i}. Then the ant chooses the task to be scheduled by using the probability distribution (over the set of unscheduled tasks):

$$prob_k^a(i) = \frac{\eta_k(i, p_{best}^i)^\alpha \tau(i, p_{best}^i)^\beta}{\sum_{i \in U_k} \eta_k(i, p_{best}^i)^\alpha \tau(i, p_{best}^i)^\beta} \quad (4)$$

where U_k denotes the list of unscheduled tasks and $\alpha > 0$, $\beta > 0$ are constants controlling the influence of pheromone value (τ) and of the local cost of pair (i, j):

$$\eta_k(i, j) = \frac{1}{T(j, S, k-1) + ET(i, j)} \quad (5)$$

In this equation, $T(j, S, k-1)$ denotes the time moment when the processor j becomes free in the context of the first $k-1$ steps of the schedule S partially constructed by the ant.

The pheromone value $\tau(i, j)$ is adjusted after each epoch, e (an epoch correspond to the construction of an entire schedule by each ant) by using the following rule:

$$\tau(i, j) = \begin{cases} \rho\tau(i, j) + \dfrac{1-\rho}{ms^*(e)} & if\ (i, j) \in S^*(e) \\ \rho\tau(i, j) & otherwise \end{cases} \quad (5)$$

where ρ is related to the pheromone evaporation rate and $ms^*(e)$ denotes the makespan of the best schedule $S^*(e)$ constructed at the previous epoch (e). At each epoch, each ant makes n steps and at each step k a task i_k is selected. This selection can be made either by generating i_k according to the probability distribution given in Eq. (4) or by choosing the task corresponding to the maximal probability. However in this second case if all ants start from the same node they will generate the same schedule.

A compromise solution is to choose i_k having the maximal value of $prob_k^a(i)$ with a probability q and to generate i_k according to the probability distribution given in Eq. (4) with probability $1-q$.

C. The rebalancing mechanism

Following the conclusions of previous studies [12] that local search can significantly improve the behavior of an ant-based scheduler we applied an improvement step to the best schedule found at each epoch. This improvement step is based on a rebalancing operator.

More specifically, in the rebalancing step the same actions take place like in case of mutation for the evolutionary scheduler. This operation is repeated for a given number of times or until no improvement can be obtained.

D. Adaptation to the changes into environment

In the case of a dynamic environment the scheduling process consists of consecutive scheduling events. At each scheduling event there is a list of available processors which is partially different from the list corresponding to the previous step. If the difference between the list of available processors is not too large then one can exploit the schedules constructed at the previous scheduling event. The pheromone matrix ensures the communication between ants being shared by all ants at each epoch of a scheduling event. Thus it seems natural to use it also for communication between the scheduling events. This means that when a schedule should be constructed for a new list of available processors the pheromone matrix is not randomly initialized but the values computed at the previous scheduling event are used. The main particularity of this approach is that the pheromone values corresponding to unavailable processors are just kept unchanged during the construction of a new schedule. Once a processor become available again their corresponding values in the pheromone matrix become active again and are transformed by the typical evaporation and reinforcement operations.

VI. EXPERIMENTAL RESULTS

In the experimental studies we analyzed the behavior of both the evolutionary scheduler (ES) and the ant scheduler (AS) in the context of both static and dynamic environments. The main aim was to identify elements of these approaches which ensure a good behavior of the schedulers especially in the case of dynamic environments.

A. Experimental setup

The test data we used are from the benchmark introduced in [5] which provides matrices containing values of the expected time for computation (ET) generated based on different assumptions related to task heterogeneity, resource heterogeneity and consistency. The heterogeneity refers to the variance among the execution times of tasks and processors. With respect to this criterion there are two main classes: low and high heterogeneity. The benchmark contains combinations of these classes both for tasks and for processors. In our analysis we used the data characterized by highly heterogeneous tasks and processors. With respect to the consistency there are three main types of ET matrices: consistent, semi-consistent and inconsistent. The consistent matrices correspond to the case when if a processor, p_i, executes a task faster than another processor, p_j, then all tasks are executed faster on processor p_i than on processor p_j. The consistent matrices model heterogeneous systems where the processors vary only with respect to their speed. In the

inconsistent case the processor p_i can execute some tasks faster than p_j and other ones slower, meaning that the environment is heterogeneous not only with respect to the speed of processors but also with respect to other characteristics which are differently exploited by different types of tasks (e.g. performances of floating point arithmetic). Semi-consistent matrices contain consistent submatrices of given sizes and models environments consisting of sub-networks of processors which are different only with respect to their speed. In our experimental study we used the matrices corresponding to a high heterogeneous environment corresponding to 512 tasks and 16 processors. Different levels of consistency were considered, thus we used the first files from classes "u_c_hihi" (consistent computing environment), "u_i_hihi" (inconsistent computing environment), "u_s_hihi" (semi-consistent computing environment).

TABLE I. Average percent of the differences between the lists of available machines corresponding to consecutive scheduling events

Unavailable machines (%)	Average (%)	Standard deviation (%)
10	12	2
20	28	8
40	48	11

In order to simulate the dynamic character of the environment we generated at each scheduling event a new list of available processors by just randomly removing a given percent of processors from the initial list of 16 resources. The percent of unavailable processors we used in our experiments are 10%, 20% and 40%. The difference in the lists of available processors corresponding to consecutive scheduling events varies both with the above mentioned percent and with the scheduling event. The average value of the number of differences between lists of available processors at consecutive steps is presented in Table I. For a sequence of 50 scheduling events Figure 1 illustrates for each event the number of differences between the current list of available processors and the previous list. The value of this difference varies between 0 (when the percent of removed processors is 10%) and 12 (when the percent of removed processors is 40%).

B. Comparative results in the static case

Before analyzing the behavior of the evolutionary and ant schedulers in the simulated dynamic environment we studied their ability to construct schedules in the static case for three types of environments: consistent (C), semi-consistent (S) and inconsistent (I). The results presented in Table II were obtained for the Evolutionary Scheduler (ES) described in Algorithm 1 and the Ant Scheduler (AS) described in Algorithm 3. In the case of ES the population consists of 200 elements, the number of generations was set to 1000 (similar to the values used in [7]), the mutation probability was set to $p_m = 0.9$.

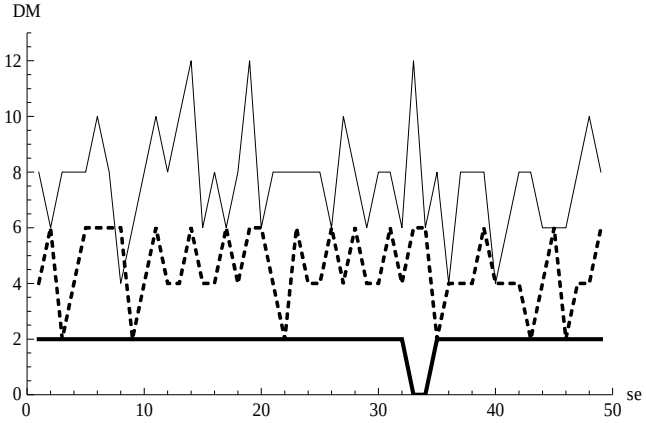

Fig. 1. The number of differences (*DM)* between the lists of available machines at two consecutive scheduling events (*se*). The lists of available machines (processors) is generated at each scheduling event by marking as unavailable a given percent from the total list of 16 machines. The used percents are: 10% (thick line), 20% (dashed line), 40% (normal line).

A preliminary parameter study suggested that the probability, p_R, of selecting the "rebalancing"-mutation has an influence on the quality of generated schedules but it should be correlated with the consistency degree of the environment. Therefore the results presented in Table II for ES were obtained for $p_R = 1$ in the case of a consistent problem, and for $p_R = 0.9$ for semi-consistent and inconsistent problems. In the case of inconsistent problems even smaller values of p_R (e.g. $p_R = 0.5$, $p_R = 0.25$) led to good results, suggesting that in this case the "shift"-mutation can improve the quality of the schedule. The probability of random perturbation (p_R in Algorithm 1) was set to 0.25.

In the case of the Ant Scheduler we used 10 ants and 500 generations. The number of rebalancing steps used to improve the quality of the schedule constructed by the best ant was set to 20. The parameter which allowed us to control the AS behavior was the probability q used in selecting the task i_k to be scheduled at step k. For consistent and semi-consistent problems the best result was obtained for $q = 0.99$ while for the inconsistent case it was obtained for $q = 0.5$.

TABLE II. The best makespan obtained by different approaches

Pb.	MinMin	GA-s [7]	ES	AS
C	8460675	7752349.4	7768564.9 ± 30862.5	8263303.8 ± 258165.7
S	5160342	4371324	4378154.7 ± 28627.8	4669991 ± 58607.2
I	3513919	3080025.8	3020607.2 ± 40757.1	3023847.1 ± 15283.6

The results presented in Table II are obtained by 30 independent runs. This table also contains results obtained for the same problems by the heuristic MinMin and the

genetic algorithm (GA) proposed in [7]. The results obtained by GA [7] are slightly better than those obtained by our ES and AS in the case of consistent and semi-consistent problems but, since the standard deviation values for GA are not provided is hard to evaluate the statistical significance of this difference. On the other hand both ES and AS behave better in the inconsistent case.

TABLE III. Ratio of scheduling events when there are statistically significant differences between the dynamic and static variants of the ES and AS.

Pb.	Unavailable machines (%)	ES		AS	
		Dyna-mic	Static	Dyna-mic	Static
C	10	*42/50*	0/50	0/50	*49/50*
C	20	5/50	3/50	2/50	*39/50*
C	40	1/50	3/50	0/50	*30/50*
S	10	*35/50*	1/50	0/50	*49/50*
S	20	5/50	2/50	5/50	*43/50*
S	40	0/50	3/50	12/50	*35/50*
I	10	*31/50*	0/50	*49/50*	0/50
I	20	4/50	1/50	*39/50*	1/50
I	40	0/50	1/50	*20/50*	2/50

C. Comparative results in the dynamic case

In the case of the simulated dynamic environment we analyzed the behavior of the dynamic ES described in Algorithm 2 and of the dynamic AS described in Algorithm 4. In both cases the number of generations (epochs) corresponding to one scheduling event was set to 50. For ES the population size was set to 512 while the number of ants was set to 10 .

The aim of these experiments was to analyze the impact of the memory mechanisms (use of previous schedules or previous values of the pheromones) on the ability of the evolutionary and ant schedulers to adapt to dynamic environments. Therefore we computed the ratio between the makespan of the schedules obtained by the static variant of the algorithm (which do not use memory mechanisms) and the makespan of schedules generated by dynamic variants. Values higher than 1 for this ratio suggest that the used memory mechanisms are effective. In the case of the evolutionary scheduler we analyzed the two variants of using information from previous scheduling steps presented in subsection IV.E. Better results were obtained in the case when an archive was used, therefore all results presented in this subsection correspond to this case. In the case of the ant based scheduler the memory mechanism consists in preserving the pheromone matrix from one scheduling event to the next one and in ”freezing” the pheromone value corresponding to resources which are not available during the current scheduling event.

Table III contains the ratio of scheduling events when the dynamic variants outperformed the static variants and also the ratio of events when the static ones outperformed the dynamic ones. In all other cases the static and dynamic variants behaved similarly. The decision that a variant outperformed the other one was taken based on comparing the averages (computed over 30 independent runs) of the makespan by using a statistical t-test (with a level of significance of 0.05). The main remark is that the behavior of the two schedulers is different. In the ES case as long as the difference between the lists of available machines corresponding to consecutive events is in average at most 12% (see Table I and Figure 2) then using then using the archive of previous schedules is beneficial for all environment types (consistent, semi-consistent, inconsistent). However the benefit of using a memory mechanism is larger in the case of consistent environments (Figure 2) and smaller in the case of inconsistent ones (Figure 4). In the case of the ant-based scheduler the memory mechanism based on the preservation of the pheromone matrix does not lead to improvements over the static AS variant in the case of consistent environments (see Table III and Figure 3). On the other hand in the case of an inconsistent environment the dynamic AS variant does not only have a better behavior than the static AS but it also leads to schedules better than those obtained by ES (see Figure 4).

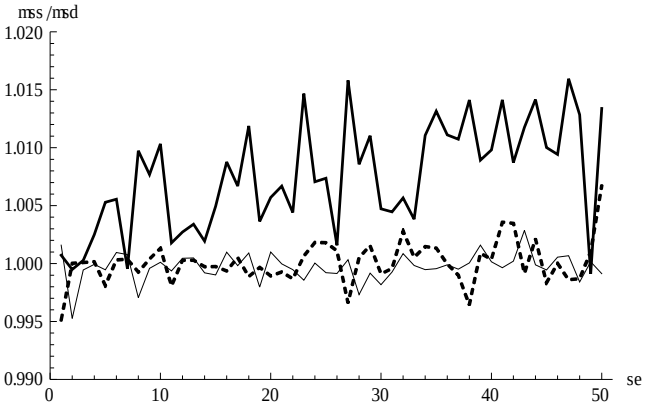

Fig. 2. The dependence between the ratio mss/msd *(mss is the makespan obtained by the static ES and* msd *the makespan obtained by the dynamic ES) and the scheduling event (*se*). Test case: consistent behavior of machines ("u_c_hihi.0"). Percent of unavailable machines: 10% (thick line), 20% (dashed line), 40% (normal line).*

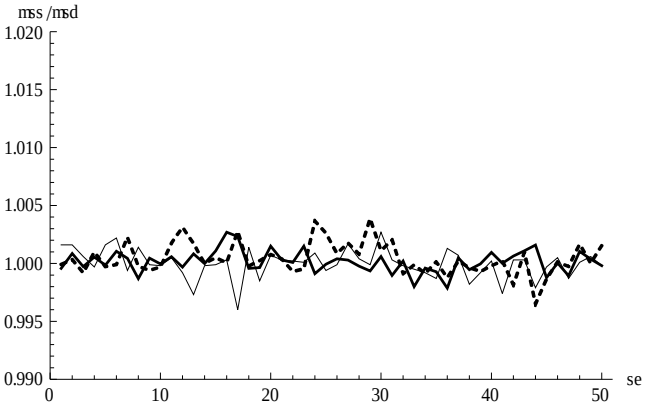

Fig. 3. The dependence between the ratio mss/msd *(mss is the makespan obtained by the static AS and* msd *the makespan obtained by the dynamic AS) and the scheduling event (*se*). Test case: consistent behavior of machines ("u_c_hihi.0"). Percent of unavailable machines: 10% (thick line), 20% (dashed line), 40% (normal line).*

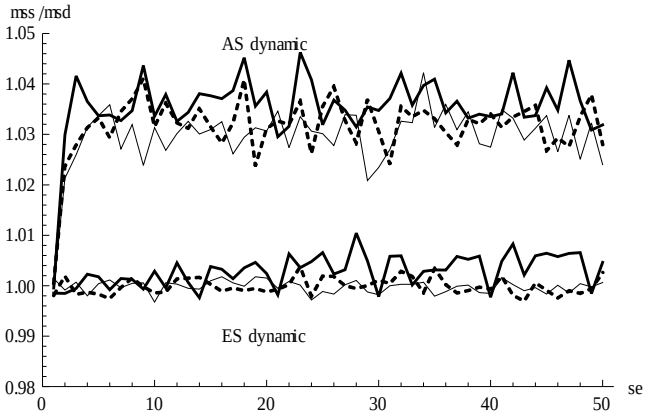

Fig. 4. The dependence between the ratio mss/msd *(*mss *is the makespan obtained by the static versions and* msd *is the makespan obtained by the dynamic versions of the algorithms) and the scheduling event (*se*). Test case: inconsistent behavior of machines ("u_i_hihi.0"). Percent of unavailable machines: 10% (thick line), 20% (dashed line), 40% (normal line).*

D. Results of a robustness analysis

There are three properties a good schedule should satisfy: (i) it should be obtained in a small amount of time; (ii) it should be easily adapted to a slightly different context; (iii) it should be robust meaning that it is affected as little as possible by run time changes [18]. There are at least two approaches in ensuring the schedules robustness. The first one is based on overestimating the execution times (this variant may induce idle times) and the other one is based on rescheduling the tasks dynamically (in this case it is important to start from a good schedule which is at least partially robust).

When analyzing the ability of an algorithm to generate robust schedule there are a few steps to be followed [18]: (i) choose a performance metric (e.g. makespan); (ii) identify the parameters which make the performance metric uncertain (e.g. the estimation of the execution times); (iii) find how a modification of these parameters changes the values of the performance metric; (iv) identify the smallest variation of a parameter that makes the performance metric to exceed an acceptable bound. A simple robustness analysis is can be conducted by firstly construct a schedule using the current estimates of the execution times, then generate a large set of perturbed values of the execution times and recompute the makespan of previously constructed schedules using the perturbed execution times.

In our experiments we generated a set of 20000 perturbed execution times by using the following random perturbation:

$$ET'(i,j) = ET(i,j)(1+\xi) \quad (7)$$

where ξ is a random value uniformly generated in $[-b,b]$ with $b \in \{0.1, 0.25, 0.5, 0.75\}$.

The dependence of the average makespans on the noise bound computed for the perturbed execution times in the case of schedules generated by the classical MinMin heuristic and the nature-inspired metaheuristics presented in this paper (ES and AS) is illustrated in Figure 5 (for consistent environments) and Figure 6 (for inconsistent environments). The obtained results suggest that for consistent environments the MinMin heuristics leads to the most robuts schedules while in the case of inconsistent environments the ES and AS schedulers lead to more robust solutions.

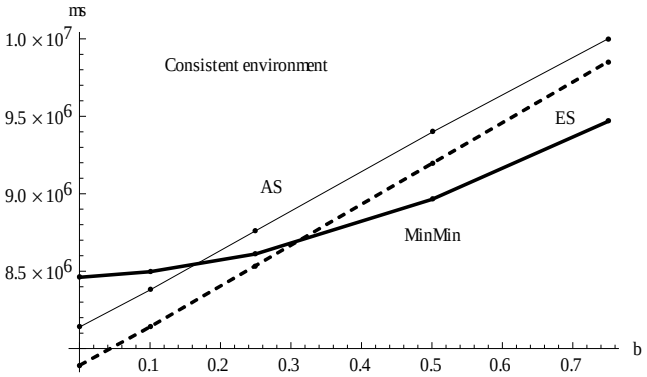

Fig. 5. Robustness analysis results for consistent environments: averaged makespan (ms) versus noise bound (b).Analyzed schedulers: MinMin (thick line), ES(dashed line), AS(normal line)

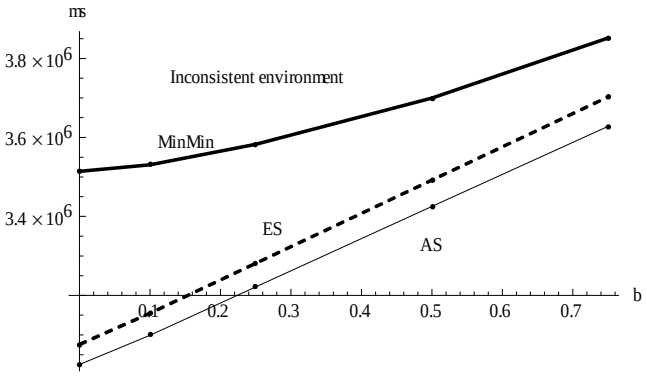

Fig. 6. Robustness analysis results for inconsistent environments: averaged makespan (ms) versus noise bound (b).Analyzed schedulers: MinMin (thick line), ES(dashed line), AS(normal line)

VII. CONCLUSIONS

The experiments conducted in the static context suggest that both in the case of the evolutionary scheduler and in the case of the ant-based scheduler there are two elements which have a beneficial impact on the quality of the generated schedules: the use of heuristic in the initialization phase and the use of a ”rebalancing” operator. In the case of ES the initial population contains, besides randomly generated schedules, also schedules fully or partially constructed using well-known heuristics (e.g. MinMin) while in the case of AS in the initial pheromone matrix the

elements corresponding to the MinMin heuristic are reinforced. On the other hand the ”rebalancing” operator based on the idea of moving tasks from the processors with high load to processors ensuring a smaller makespan proved to be beneficial for ES especially in the case of consistent environments. For inconsistent environments less greedy operators (e.g. the ”shift” operator) proved to help the construction of good schedules. For the ant-based scheduler the rebalancing of the schedules constructed by the best ant also helped in improving its quality. An important element for the behavior of ES was the random perturbation used in the selection step which allows the evolutionary process to preserve the population diversity and avoid premature convergence. The behavior of ES in the static case is comparable with the behavior of others schedulers based on evolutionary algorithms [5], [7]. The use of information collected from previous scheduling steps helps both ES and AS to adapt to the environment. For ES the percent of elements preserved from previous population(s) should not be larger than 10% in order to lead to a gain in the quality of evolved schedules with respect to the static variant. The gain of the dynamic variants over the static ones is significant if the differences between the lists of available processors corresponding to consecutive scheduling events are smaller than 20%. However in the case of inconsistent environments the ant-based scheduler with pheromone matrix transferred from one scheduling step to another leads to better results than the static AS even if the difference between the lists of available processors is around 40%. In case of consistent and semiconsistent data the AS in dynamic case does not perform well but it seems to improve while the number of unavailable processors increases.

A similar situation holds also with respect to the robustness of the generated schedules. The schedules generated by ES and AS are significantly more robust than the schedules generated by the MinMin heuristics in the case of inconsistent environments. On the other hand the MinMin heuristics lead to more robust schedules in the case of consistent environments.

ACKNOWLEDGMENTS

This work was supported by the Romanian grant PN-II 11-028/ 14.09.2007.

REFERENCES

[1] A. Abraham, H. Liu, C. Grosan,F. Xhafa, Nature Inspired Meta-heuristics for Grid Scheduling: Single and Multi-objective Optimization Approaches, F. Xhafa, A. Abraham (Eds.): Meta. for Sched. in Distri. Comp. Envi., SCI 146, pp. 247-272, 2008.

[2] F. Xhafa, A. Abraham, Meta-heuristics for Grid Scheduling Problems, Metaheuristics for Scheduling: Distributed Computing Environments, Studies in Computational Intelligence, Springer Verlag, Germany, pp. 1-37, 2008.

[3] A.M. Mehta, J. Smith, H.J. Siegel, A. Maciejewski, A. Jayaseelan, B. Ye, Dynamic Resource Allocation Heuristics for Maximizing Robustness with an Overall Makespan in an Uncertain Environment, Proceedings of PDPTA 2006, Las Vegas, Nevada, USA, June 26-29, 2006, vol. 1, 2006.

[4] F. Zamfirache, D. Zaharie, C. Craciun, Evolutionary Task Scheduling in Static and Dynamic Environments, Proc. ICCC-CONTI, Timisoara, Romania, 2010.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61(6), pp. 810-837, 2001.

[6] F.Xhafa, A. Abraham, Computational models and heuristic methods for Grid scheduling, Future Generation Computer Systems, 26, pp. 608-621, 2010.

[7] J. Carretero, F. Xhafa, Using Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications. Journal of Technological and Economic Development - A Research Journal of Vilnius Gediminas Technical University, 12(1), pp. 11-17, 2006.

[8] B. Sahoo, S. Mohapatra, and S.K. Jena, A Genetic Algorithm Based Dynamic Load Balancing Scheme for Heterogeneous Distributed Systems, Proceedings of PDPTA 2008, Las Vegas, Nevada, USA, July 14-17, 2008, CSREA Press, 2008.

[9] G. V. Iordache, M. S. Boboila, F. Pop, C. Stratan, V. Cristea, A Decentralized Strategy for Genetic Scheduling in Heterogeneous Environments. OTM Conferences (2) , pp. 1234-1251, 2006.

[10] R. Prodan, T. Fahringer, Dynamic scheduling of scientific workflow applications on the grid: a case study. Procs. of ACM SAC 2005, pp. 687-694, 2005.

[11] S. Fidanova, M. Durchova, Ant Algorithm for Grid Scheduling Problem. In LNCS 3743, pp. 405-412, 2006.

[12] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In Proceedings the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, ISSN 1368-5708, 2004.

[13] K. Rzadca, F. Seredynski, Heterogeneous multiprocessor scheduling with differential evolution, Congress on Evolutionary Computation IEEE (2005) , pp. 2840-2847,2005.

[14] P. Visalakshi, S. N. Sivanandam, Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization, Int. J. Open Problems Compt. Math., 2(3), 2009.

[15] F. Xhafa, B. Duran, A. Abraham, K. Dahal, Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids, Neural Network World, 18(3), pp. 209-225, 2008.

[16] K. Kousalya, P. Balasubramanie, An enhanced ant algorithm for grid scheduling problem. In IJCSNS International Journal of Computer Science and Network Security, vol. 8, no. 4, 2008.

[17] S. Lorpunmanee, M.N. Sap, A.H. Abdulah, C. Chompooinwai, An ant colony optimization for dynamic job scheduling in grid environment. In International Journal of Computer and Information Science and Engineering vol. 1, no. 4, pp. 207-214, 2007.

[18] L.C. Canon, E.Jeannot, R. Sakkellariou and W.Zhang, Comparative evaluation of the robustness of DAG scheduling heuristics, Grid Computing, pp. 73-84, 2008.