# Multiprocessor Scheduling with Evolving Cellular Automata Based on Ant Colony Optimization

Toktam Ghafarian
*Ferdowsi University of Mashhad , Department of Computer Engineering , Mashhad , Iran.*
*Khayam Higher Education Institute, Mashhad , Iran*
*ghafarian@stu-mail.um.ac.ir*

Hossein Deldari
*Ferdowsi University of Mashhad, Department of Computer Engineering , Mashhad , Iran.*
*hd@ferdowsi.um.ac.ir*

Mohammad-R. Akbarzadeh –T.
*Ferdowsi University of Mashhad, Department of Computer Engineering and Electrical Engineering, Mashhad ,Iran.*
*akbarzadeh@ieee.org*

## Abstract

*Multiprocessor scheduling belongs to a special category of NP-complete computational problems. The purpose of scheduling is to scatter tasks among the processors in such a way that the precedence constraints between tasks are kept, and the total execution time is minimized. Cellular automata (CA) can be used for multiprocessor scheduling, but one of the difficulties in using CA is the exponentially increasing number of rules with increasing number of processor and neighborhood radius. Here, we propose a combined use of ant colony and evolutionary meta-heuristics to search the rule's feasible space in order to find optimal rule base. Also we introduce a two dimensional cellular automata structure based on the important task attributes in the precedence task graph. The proposed scheduler that uses evolving cellular automata based on ant colony can find optimal response time for some of well known precedence task graph in the multiprocessor scheduling area.*

## 1. Introduction

Over the years, cellular automata (CA) has been used to model many complex dynamic systems in discrete space and time. It contains an array of cells with a finite number of possible states. The cells are updated synchronously or asynchronously in discrete time steps according to its current states and those of its neighbors [12, 13]. Designing a CA which performs a particular task is highly complicated, thus it limits their applications, but the automatic design enhances the viability of CA. The Evolving Cellular Automata (EvCA) system [4] uses genetic algorithms (GA) to search the space of possible rules in order to find optimal rule base. In this paper, we propose a two dimensional evolving CA based on ant colony optimization method. This framework is used to find optimal response time of scheduling parallel tasks in the homogeneous multiprocessor system. The extended version of his solution can be used in the scheduler of the real grid system .Parallel task scheduling with precedence constraints is a popular problem and a number of heuristics have been proposed [3, 6, 7, 8, 9, 10] to date.

The remainder of this paper is organized as follows. Section 2 presents the scheduling problem. Section 3 presents the concept of multiprocessor scheduling with the use of CA, Section 4 discusses evolving CA based on ant colony, and finally section 5 contains experimental results.

## 2. Multiprocessor Scheduling Problem

The precedence task graph is one of the most popular parallel application models. In this model, an application can be displayed by a Directed Acyclic Graph (DAG) in which nodes display atomic tasks and the directed edges represent the execution dependencies between nodes. The weight of each node represents the computation time associated with a task; while the number next to each edge denotes the communication time between the source and sink nodes.

A node with no parent is called an entry node, while a node without any child is called an exit node. Fig. 1 shows an example of the precedence task graph with six tasks. The purpose of the scheduling problem is mapping tasks to the processors in such a way that the total execution time is minimized.

We represent some definition for precedence task graph in Table 1.Also In this paper, we use some definition defined as follows:

the level of node $n_i$ represented by $level\ (n_i)$ is recursively defined as follows:

$level(n_i) = w(n_i)$ For an exit node (1)

$level(n_i) = \max_{l \in parent(n_i)} (level(n_l) + c_{il}) + w(n_i)$
for other nodes

We can compute static level of a given task before assigning a task in a particular processor and compute dynamic level after assigning a task to a particular processor of a parallel system.The absolute earliest start time of a node $n_i$ , denoted by $AEST(n_i)$ is recursively defined as follows:

$AEST(n_i) = 0$ If it is an entry node
$AEST(n_i) = \max_{1 \le k \le p} AEST(n_{i_k}) + w(n_{i_k})$
$+ r(PE(n_{i_k}), PE(n_i))c_{i_k i}$
Where $n_i$ has p parent nodes and $n_{i_k}$ is the kth parent node. (2)
$r(PE(n_{i_k}), PE(n_i)) = 1$ If $n_i$ and $n_{i_k}$ run on different processors and zero otherwise

We can obtain AEST value by traversing the task graph in the breath first search starting from the entry nodes. the dynamic critical path length, denoted by DCPL, is defined as:

$$\max_i \{AEST(n_i) + w(n_i)\} \quad (3)$$

The value of DCPL simply represents the schedule length of the partially scheduled task graph. Because it is computed by taking the maximum value across all the earliest finish time.

TABLE I. SOME NOTATION OF THE PRECEDENCE TASK GRAPH

| Symbol | Meaning |
|---|---|
| $n_i$ | The node number of a task in parallel program task graph |
| $w(n_i)$ | The computation cost of node $n_i$ |
| $c_{ij}$ | The communication cost of the directed graph from $n_i$ to $n_j$ |
| $PE(n_i)$ | The processor that contain node $n_i$ |

The absolute latest start time of a node $n_i$ , denoted by $ALST(n_i)$ is recursively defined as follows:
$ALST\ (n_i) = DCPL - w(n_i)$
If it is an exit node

$ALST(n_i) = \min_{1 \le m \le q} \{ALST(n_{i_m}) - r(PE(n_{i_m})$
$, PE(n_i))c_{ii_m} - w(n_i)\}$
*Where $n_i$ has q children nodes and $n_{i_m}$ is the mth child node.* (4)

$r(PE(n_{i_m}), PE(n_i)) = 1$ If $n_i$ and $n_{i_m}$ run on different processors and zero otherwise

The values of ALST can be computed by traversing the task graph in a breadth-first search in the reverse direction. With each node assigned AEST and ALST, the node on the DCP can be easily recognized.

$$AEST(n_i) = ALST(n_i) \quad (5)$$

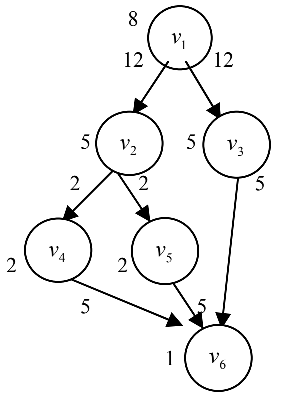

Figure 1. an example of the precedence task graph

## 3. Multiprocessor scheduling with Evolving Cellular Automata

We assume there are two processors in the system ($p_0, p_1$). In this paper, a program graph is modeled with a DAG. We define irregular two dimensional binary CA corresponding to the each graph. Each cell of CA is associated with a task of the program graph. The state of each cell determines the processor on which a task should run. We define particular selected neighborhood for each cell. First of all, we determine critical parents and critical children for each task of the program graph. Critical parent (child) is a parent (child) that has smallest differences between ALST and AEST refer to "(4)", "(2)". Critical parents and critical children take into account as neighbors of each cell in CA. in this way, selecting the processor for each task bound to the processors that run critical parents and critical children thus it helps reduce the start time of each task [8].

Initially, each task (cell) is randomly assigned to the processor. Since there is irregularity in DAG we have some special cases that are explained below:

1. If the task doesn't have parent node such as entry node or doesn't have any children nodes such as an exit node, we add two dummy nodes with dummy state -1 like $v_1, v_6$ in Fig. 2.
2. If the task only has one parent or child, we add one dummy node with the state of corresponding parent or child like the nodes $v_2, v_3, v_4, v_5$ in Fig. 2.
3. If the task has more than two parent or child, we select two critical parents or children like $v_6$ in Fig. 2 that has 3 parent nodes.

After we define the structure of CA , we should define a general rule for CA. the state of central cell can have two values ({0,1}) and the state of sub neighborhood takes 5 values ({0,0} { 0,1} {1,0} {1,1} {-1,-1}) thus the total number of possible states configuration is 2*5*5=50, so that , the length of general rule is 50 bits and thus there are possible general rule for proposed CA. in comparison with GA based CA in [1] we use shorter general rule thus the search space is smaller than GA based CA . We search this huge size of possible general rule space with ant colony optimization method.

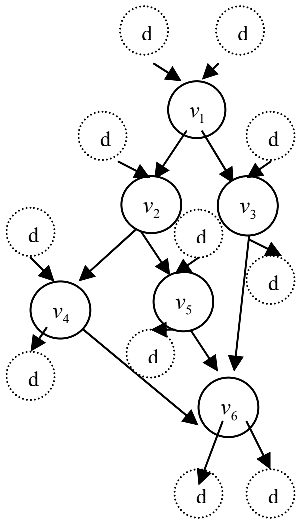

Figure 2. Selected neighborhood of each task that is represented with DAG in Fig. 1

## 4. Evolving CA Based on Ant Colony Optimization Method

In this paper, the search space of CA general rule explore by an ant colony optimization method (ACO). In the learning phase, we test two type of ACO, Ant Colony System (ACS) introduced by Dorigo and Gambardella (1997) [11] and MAX-MIN Ant System (MMAS) proposed by Stützle and Hoos (2000) [14]. Since , the feasible solution created by each ant is binary string , we use a special instance of the ACS and MMAS meta heuristic that is called HCF-ACO especially designed to deal with binary problem [5].

Initially each ant start your trip from a random point in the general rule .Each ant follow a simple heuristic method that guide it in the search space. Suppose an ant start your trip from 16 th location in general rule , in this location we have this configuration 01000. from left , subsection 01 represent the processors which run critical parents , subsection 0 indicate the processors that run task itself and subsection 00 display the processors for critical children. this configuration states that one of critical parent run on processor 1 and the other one run on processor 0 (subsection 01) , but both of critical children run on processor 0 (subsection 00) , in this case heuristic function that guide ant in the search space prefer to select processor 0 , because two critical children and one of the critical parent run on processor 0 thus running the task on processor 0 decrease the start time of the task and therefore total execution time .

Certainly heuristic function , prefer to select the processor of critical parents rather than critical children unless the parents run on two processors like our example, therefore in this situation it prefers the processor of critical children. If critical children also run on two processors, it selects target processor randomly. . At each step of trip, an ant constructs a portion of general rule (one bit) and continue to make until its trip is completed.

After each ant construct individual general rule, we should determine the fitness of these general rules. In the evolving CA based on GA , the fitness of general rule can be computed via some test problem [1, 2, 4]. Therefore we create a set of test problem , each test problem represent initial random assignment of tasks in the desired graph to processors . At first, CA is initialized according to the first test problem and equipped with the general rule selected from the population of general rules, and start to evolve; changing its state during number of predefined step, a final allocation of tasks to processors can use to determine the final schedule length of a desired graph. After CA determine witch task run on witch processor, we use the local scheduling policy on each processor. For a given rule, this above procedure is repeated for the certain number of test problem. The fitness value of selected general rule is computed via the average of values T corresponding to the individual run. Notice that all general rules are tested on the same set of test problem, but these test set is different from generation to generation in ACO. We use two heuristic algorithms for local scheduling on each machine. DCP heuristic Algorithm [8] in which we assign dynamic priority to the tasks based on dynamic critical path refer to "(5)" , thus the task with smaller differences between ALST,AEST has higher priority than the other tasks . The other heuristic algorithm is based on highest static or dynamic level first refers to the "(1)". Evolving CA scheduler based on ACO is summarized below:

```
Create an initial population of m ant.
  While number of generation < max number of generation of ACO
      Create a set of n test problems
    Each ant start your trip from a random position (bit) of CA general rule
    do
    Each ant create a portion of trip (a bit) according to the heuristic method and pheromone trail
      Do local pheromone update phase (this phase only does in ACS)
  While each ant complete its trip        //end do-while loop
// this section compute the fitness of general rule
 For each of m ant and each of n test problem
        $T_{mn}^* = T_{mn}^* + CA(rule_m, test_n, CA - step)$
        (CA initialize with n^rd test problem and equip with the general rule of m^rd ant and start to evolve for CA-step times then it return the total execution time of desired graph as $T_{mn}^*$ )
        $fitness_m = T^*_{mn} / n$   (The average execution time of the general rule created by the m^rd ant)
 End // end loop
        Do pheromone update phase
   End //end while loop
```

## 5. Experimental Result

In the experiments reported in this section , we assume that the CA works asynchronously so that only one cell update it's state at a given time step and a single step of
CA is completed in a number of tasks .

The number of time step for running CA is equal to two times of the cell. A number of experiments with the popular program graph in the scheduling area have been conducted. The first program graph referred as gauss18 is displayed in Fig. 3(a) .It showed the parallel Gaussian elimination Algorithm that made of 18 tasks. The next program graph g18 present in Fig. 3(b).

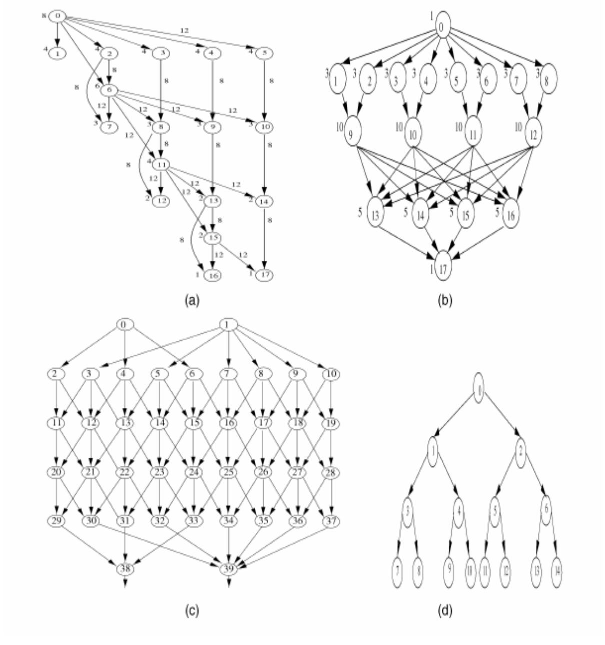

Figure 3. Program graphs: (a) gauss18, (b) g18, (c) g40, and (d) tree15. Figures are taken from [2]

Computational costs of tasks are represented in the figure and communication costs are all the same and equal to 1. Fig. 3(c) shows a program graph g40 with computational and communication costs equal to 4 and

1, respectively. Fig. 3(d) displays a binary out-tree program graph referred as tree15. Computation and communication weights of out-trees are equal to 1. in our experiments we use these parameters : $\alpha = 2, \beta = 1, \rho = 0.1$

## 5.1 program graph Gauss18

In this experiment, ACO type is MMAS, we have 100 ant and number of generation equal to 100. local scheduling policy on each processor based on DCP method, In this experiment, according to the [15] we alternate the pheromone trail update between $s^{gb}$ and $s^{ib}$ thus gradually shifting the emphasis from the iteration-best to the global-best solution for the pheromone trail update till we gain a stronger exploration of the search space early in the search and stronger exploitation later in the run. As you seen in Fig. 4 one notice that this evolving automata can get optimal response time *T=44* in early generation 15 and occur some times after. In comparison to GA based CA [1,2] optimal solution occur sooner.

## 5.2 program graph Guass18

We use ACS in this experiment with these parameters; 100 ant and number of generation is equal to 100, heuristic scheduling algorithm is based on highest dynamic level first. As Fig. 5 showed; we can get optimal response time with value *T=44*. In this experiment we conclude that changing the local scheduling algorithm on each processor doesn't impact on total response time. In comparison to GA [1,2] in this case, like previous experiment, we can get optimal solution sooner . The optimal solution occurs near to generation 10 and repeated some times after.

## 5.3 program graph g40

We use ACS with the same parameter as we use in the previous experiment. Heuristic scheduling policy is base on DCP method. As you can see on Fig. 6 we can get optimal response time *T=80* in generation 8 and some generations after. In comparison to GA [1,2] we can get optimal solution in less number of generations, in GA after 160 generation, the optimal solution appears.

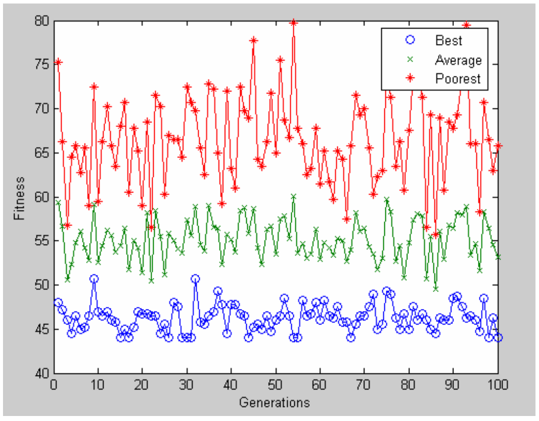

Figure 4. Response time of evolving CA with MMAS for guass18

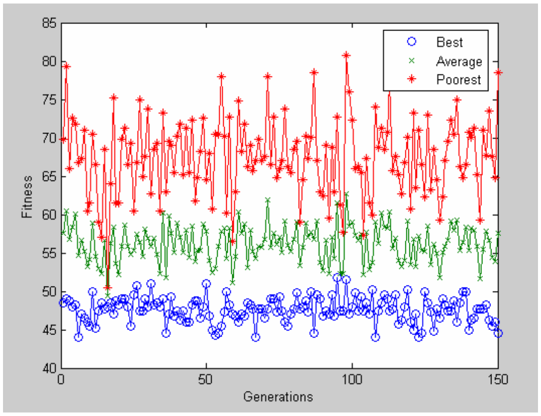

Figure 5. Response time of evolving CA with ACS for guass18 graph

## 5.4 program graph g18

We use MMAS with these parameters; 100 ants with 100 generations also the heuristic function is based on highest static level first. We show my result in Fig. 7 . We can get optimal response time *T=46* for this graph In compare to GA[1,2] the behavior of proposed algorithm is very near to the behavior of GA based CA because both of them can get optimal solution before generation 20.

## 5.5 program graph tree15

In this experiment we use ACS with 50 ant and number of generation are equal to 100 with DCP method as local scheduling policy , as you noticed in Fig. 8 we can get optimal response time *T=9* almost in generation 5 and this result repeat some times after . The behavior of the proposed algorithm almost is the same as GA. based CA in [1,2]Both of them can get optimal solution almost in generation 5.

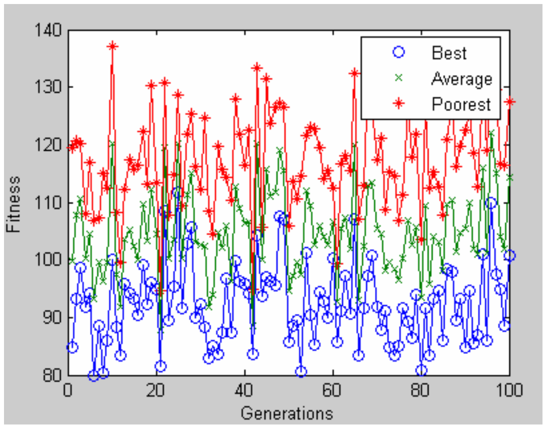

Figure 6. Response time of evolving CA with ACS for program graph g40

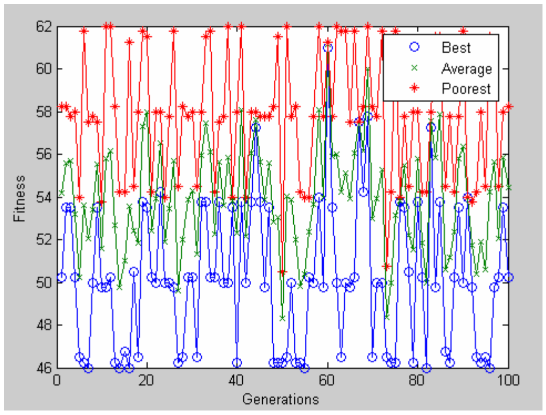

Figure 7. Response time of evolving CA with MMAS for program graph g18

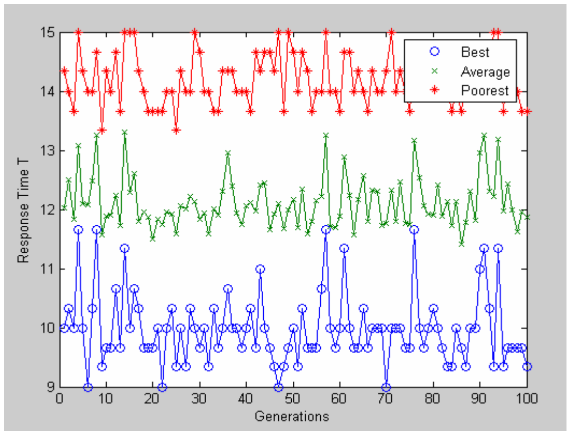

Figure 8. Response time of evolving CA with ACS for program graph tree15

## 6. References

[1] A. Swiecicka, F. Seredynski,A. Y. Zomaya, "Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support, " *IEEE trans. on parallel and distributed systems*, vol. 17, no. 3 , pp. 253-263, march 2006.

[2] F. Seredynski and A. Y. Zomaya , "Sequential and parallel cellular automata-based scheduling algorithms," *IEEE trans. on parallel and distributed systems*, vol. 13, no. 10 , pp. 1009-1023, October 2002 .

[3] G. Ritchie, "Static multi-processor scheduling with ant colony optimization and local search," Master of Science thesis, University of Edinburgh, 2003.

[4 ] J.P. Crutchfield, M. Mitchell, R. Das, Evolving Cellular Automata to Perform Computations, Handbook of Evolutionary Computation, Oxford University Press, 1997.

[5] C.Blum and M.dorigo , "The hyper-cube framework for ant colony optimization ," *IEEE Trans. on Systems , Man, and Cybernetics*, Part B , vol. 34, no. 2 , pp. 1161-1172 , 2004

[6] T. D. Braun and "*et al", "*a Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems , "Proceedings of the Eighth Heterogeneous Computing Workshop,1999.

[7] Y.-K. Kwok, I. Ahmad," Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing* , vol. 31, no. 4, pp. 406–471, 1999.

[8] Y.-K. Kwok, I. Ahmad ,"Dynamic critical–path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Trans. of Parallel and Distributed Systems*, vol. 7, no. 5 , pp. 506-521, May 1996.

[9] Y.-K. Kwok, I. Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms, "*Journal of Parallel Distributed Computation*, vol. 59, no. 3, pp. 381–422, 1999.

[10] Z. Shi ,"Scheduling tasks with precedence constraints on heterogeneous distributed computing systems," PhD thesis, The University of Tennessee, December 2006.

[11] M. Dorigo, L. M. Gambardella, "Ant colony system :a cooperative learning approach to the traveling salesman problem," *IEEE Trans. on Evolutionary Computation*, vol.1 , no. 1, 1997.

[12] S. Wolfram: Theory and Applications of Cellular Automata, Singapore, 1986.

[13] T. To_oli, N. Margolus: Cellular Automata Machines: a New Environment for Modeling ,Cambridge: MIT Press, 1987.

[14] T. Stutzle, H. H. Hoos, " Max_Min Ant System, " *Elsevier Science* , November 1999.