International Journal of Grid Distribution Computing
Vol.8, No.1 (2015), pp.41-54
http://dx.doi.org/10.14257/ijgdc.2015.8.1.05

Multiprocessor Task Graph Scheduling Using a Novel Graph-Like Learning Automata

H. R. Boveiri

Sama Technical and Vocational Training College, Islamic Azad University, Shushtar Branch, Shushtar, Iran
E-mail: boveiri {@shoushtar-samacollege.ir or @ymail.com}

Abstract

Optimized task scheduling is one of the most important challenges in multiprocessor environments such as parallel and distributed systems. In such these systems, each parallel program is decomposed into the smaller segments so-called tasks. Task execution times, precedence constrains and communication costs are modeled by using a directed acyclic graph (DAG) named task graph. The goal is to minimize the program finish-time (makespan) by means of mapping the tasks to the processor elements in such a way that precedence constrains are preserved. This problem is shown to be NP-hard in general form and some restricted ones. Therefore, utilization of heuristic and meta-heuristic approaches to solve this problem is logical. Learning automata (LA) is an abstract model to interact with stochastic environment, which tries to reform itself based on the environment feedback. Although a learning automaton itself is a simple component, a group of them by cooperating each other can show complicated behavior, and can coverage to desired solutions under appropriate learning algorithm. In this paper, an ingenious graph-like learning automata in which each task in the task graph is represented by a learning automaton tries to solve the multiprocessor task-scheduling problem in a collective manner. Set of different experiments on various real-world task-graphs has been done and archived results are so promising compared to the traditional methods and genetic algorithm.

Keywords: *Learning automata, multiprocessor task scheduling, parallel and distributed systems, task graph.*

1. Introduction

Today, utilization of multiprocessor systems has been increased due to increase in time-complexity of programs and decrease in hardware costs. In such these systems, programs are decomposed into the smaller and dependent segments named tasks. Some tasks as input need data generated by other tasks, and hence, the problem can be modeled by using a directed acyclic graph (DAG) so-called task graph. In the aforementioned graph, nodes are tasks and edges indicate precedence constraints between tasks. Required execution time of tasks, precedence constraints and communication costs are determined during compile step. Tasks must be mapped into processors with respect to their precedence in such a way that overall finish-time (Makespan) of the program can be minimized. Multiprocessor task scheduling problem is NP-hard in general form, and achieving the best possible solution is generally too time-consuming and maybe impossible especially when there are a large number of tasks in the task graph.

Different algorithms proposed to schedule multiprocessor task graph are divided into two categories, heuristics and non-heuristics. Some heuristics named TDB (Task Duplication Based) allow task duplication redundantly over processors to eliminate the communication costs between tasks and their children to make the finish time shorter such

ISSN: 2005-4262 IJGDC
Copyright ⓒ 2015 SERSC

as PY [16], DSH [15], LWB [17], BTDH [18], LCTD [19], CPFD [20], MJD [21], and DFRN [22]. However, some tasks like bank-transactions cannot be repeated.

Other heuristics do not allow task-duplication; themselves are divided into two groups. First, the UNC (Unbounded Number of Clusters) methods like LC [23], EZ [24], MD [25], DSC [26], and DSP [27], which work on unbounded number of processors using task graph clustering. That is, they part task graph into some clusters and then assign them to processors. They find optimal number of clusters (processors) implicitly. However, unlimited number of processor elements may not be available in the real problems. Second, the BNP (Bounded Number of Processors) methods like HLFET [28], ISH [15], CLANS [20], LAST [30], EFT [31], DLS [46], and MCP [25], which number of processors are restricted using list scheduling (the proposed approach is in this group). They make a list of ready tasks in each stage and assign some priorities to them. Then the task that has the most priority in the ready list is selected to schedule on the processor that allows the earliest start time, until all tasks of task graph are scheduled.

Finally, between non-heuristic approaches, one can see genetic algorithm (GA) [1]-[10], simulated annealing (SA) [33], ant colony optimization (ACO) [44], and local search [34]. Among them, genetic algorithm has significant contribution. However, necessary time to execute genetic algorithm is usually more than random running of tasks. ACO has identical performance with genetic algorithm in so lesser overload and its results are so promising.

Learning automata (LA) is an abstract model to interact with stochastic environment, which can perform finite set of actions [43]. The choice of an action depends on the state of LA represented by an action probability vector. The stochastic environment evaluates each selected action of LA, and responds to it. The LA uses this response to reform itself for the next situations. This interact is done until LA would learn to select the best action in each situation where the best action is that which has the most probability to get reward from environment. The environment is represented by triple $\mathbf{E} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{c}\}$, where $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is inputs set, $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$ is responses-set of environment and $\mathbf{c} = \{c_1, c_2, \ldots, c_r\}$ is set of probabilities of penalty of each input. $\boldsymbol{\beta}$ can be a *two*-member set so that $\beta_1 = 0$ indicate reward and $\beta_2 = 1$ is penalty. Note that values of $\mathbf{c}$ will never change in stationary environments. LA can adapt itself to environment by using an iterative learning algorithm. LA have been used successfully in many applications such as control of broadcast networks [37], intrusion detection in sensor networks [38], database systems [39], and solving shortest path problem in stochastic networks [40], [41], to mention a few.

The full potential of LA is realized when multiple automata interact with each other. Interaction may assume different forms such as tree, mesh, array, etc. In this paper, a graph of learning automata tries to solve multiprocessor task-graph scheduling in a cooperative manner. In the proposed approach, a learning automaton represents each node (task) of the task graph. An iterative algorithm is executed in which iteration makes complete scheduling. After extracting the achieved scheduling length, penalty or reward signal to feed to LA is created. Learning automata are trained to improve themselves based on desirability of achieved solution. The algorithm continues until convergence of the solutions produced by learning automata.

The rest of the manuscript is organized as follows. In the next section multiprocessor task scheduling is discussed. Section III introduces learning automata. The proposed approach is expanded in section IV. Section V devoted to implementation details and achieved results. Finally, the paper is concluded in the last section.

 Copyright ⓒ 2015 SERSC

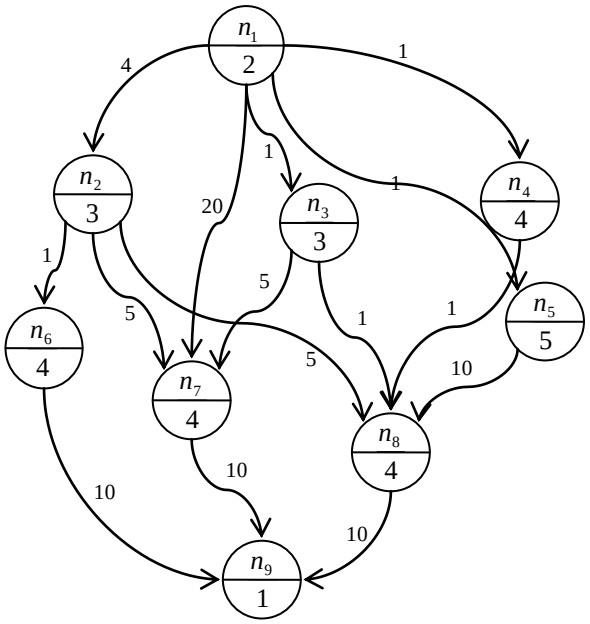


Figure 1. Task Graph of a Program with Nine Tasks [14]

2. Multiprocessor Task Scheduling

A directed acyclic graph **G** = {**N**, **E**, **W**, **C**} named task graph is used to modele a parallel program, where

N = {n_1, n_2,…, n_n},

E = {(n_i, n_j) | n_i, n_j ∈ **N**},

W = {w_1, w_2,…, w_n},

C = {c(n_i, n_j) | (n_i, n_j) ∈ **E**), and n are set of nodes, set of edges, set of weight of nodes, set of weight of edges, and number of nodes respectively.

Fig. 1 shows task graph of a program with nine tasks. In the task graph, nodes are tasks and edges specify precedence constrains between tasks. Edge (n_i, n_j) ∈ **E** demonstrate that task n_i must be finished before starting of task n_j. Each node weight w_i is the necessary execution time of task n_i and each weight of edge c(n_i, n_j) is the required time for data transmission from task n_i to task n_j identified as communication cost. If both tasks n_i and n_j are executed on the same processor then communication cost will be zero between them, else n_j must wait as c(n_i, n_j) after finishing n_i. Tasks execution times, precedence constraints between tasks and communication costs are generated during the program compile stage. Tasks must be mapped into the given m processor elements with respect to their precedence so that overall finish time of tasks would be minimized.

Most BNP scheduling algorithms are based on the so-called list-scheduling technique. The basic idea of list scheduling is to make a sequence of nodes as a list for scheduling by assigning them some priorities, and then repeatedly remove the highest priority node from the scheduling list, and allocate the node to which processor that allows the earliest-start-time (*EST*) until all nodes in the graph are scheduled. If all predecessors of task n_i are executed on the processor p_j, *EST*(n_i, p_j) will be *Avail*(p_j) that is, the earliest time that p_j is

Copyright ⓒ 2015 SERSC

available to execute next task, else the earliest-start-time of task n_i on processor p_j must be compute by using:

$$EST(n_i, p_j) = \max\left(Avail(p_j), \max_{n_m \in Parents(n_i)} \left(FT(n_m) + c(n_m, n_i) \right) \right) \quad (1)$$

where $FT(n_m) = EST(n_m) + w_m$ is actual finish-time of task n_m, and $Parents(n_i)$ is set of all parents of n_i. Finally, total finish time of program (makespan) is calculated:

$$makespan = \max_{i=1}^{n} \left(FT(n_i) \right) \quad (2)$$

Some frequently used attributes to assign priority to tasks are *TLevel* (Top Level), *BLevel* (Bottom Level), *SLevel* (Static Level), *ALAP* (As-Late-As-Possible), and the new proposed *NOO* (Number-Of-Offspring). The *TLevel* or *ASAP* (As-Soon-As-Possible) of a node n_i is length of a longest path from an entry-node to n_i excluding n_i itself, where the length of a path is the sum of all the nodes and edges weights along the path. *TLevel* of each node in task graph can be computed by traversing the graph in topological order using:

$$TLevel(n_i) = \max_{j \in Parents(n_i)} \left(TLevel(n_j) + c(n_j, n_i) + w_j \right) \quad (3)$$

The *BLevel* of a node n_i is the length of the longest path from n_i to an exit-node. It can be compute for each task by traversing the graph in reversed topological order as follow:

$$BLevel(n_i) = \max_{j \in Children(n_i)} \left(BLevel(n_j) + c(n_i, n_j) \right) + w_i \quad (4)$$

where $Children(n_i)$ is set of all children of n_i.

If we do not take into account the edges weights in computing the *BLevel*, a new attribute can be generated called Static-Level or simply *SLevel* using (5).

$$SLevel(n_i) = \max_{j \in Children(n_i)} \left(SLevel(n_j) \right) + w_i \quad (5)$$

The *ALAP* start-time of a node is a measure of how far the node's start-time can be delayed without increasing the schedule length:

$$ALAP(n_i) = \min_{j \in Children(n_i)} \left(CPL, ALAP(n_j) - c(n_i, n_j) \right) - w_i \quad (6)$$

where CPL is Critical-Path-Length that is, length of the longest path in the task graph.

The *NOO* of n_i is simply number of all its descendants. Table I lists the aforementioned measures of each node in the task graph of fig. 1. Four well-known traditional scheduling algorithms of the BNP class are surveyed as follows.

2.1 The HLFET Algorithm

The HLFET (Highest Level First with Estimated Times) [28] first calculate SLevel of each node. Then, make a ready list in a descending order of SLevel. At each instant, it schedules the first node in the ready list to a processor that allows the earliest execution time using the non-insertion approach and then, updates the ready list by inserting the nodes that are now ready. Until all nodes are scheduled. Fig. 2 (a) shows scheduling of the fig. 1 graph using HLFET algorithm on two processor elements.

 Copyright ⓒ 2015 SERSC

Table 1.
***TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* of Each Node in the Task Graph of Figure 1**

Node	*TLevel*	*BLevel*	*SLevel*	*ALAP*	*NOO*
n_1	0	37	12	0	8
n_2	6	23	8	14	4
n_3	3	23	8	14	3
n_4	3	20	9	17	2
n_5	3	30	10	7	2
n_6	10	15	5	22	1
n_7	22	15	5	22	1
n_8	18	15	5	22	1
n_9	36	1	1	36	0

2.2 The MCP Algorithm

The MCP (Modified Critical Path) algorithm [25] uses the ALAP of a node as the scheduling priority. It first computes the ALAP times of all nodes, and then constructs a ready list in an ascending order of ALAPs. Ties are broken by considering the ALAP times of the children of a node. The MCP algorithm then schedules the nodes on the list one by one to a processor that allows the earliest start time using insertion approach. Scheduling of the fig. 1 graph using MCP algorithm on two processor elements is shown by fig. 2 (b).

2.3 The DLS Algorithm

The DLS (Dynamic Level Scheduling) algorithm [46] uses an attribute called dynamic level (DL) that is the difference between the SLevel of a node and its earliest start time on a processor. At each scheduling step, the DLS algorithm computes the DL for every node in ready list on all processors. The node-processor pair that gives the largest value of DL is selected to schedule. Until all nodes are scheduled. The algorithm tends to schedule nodes in a descending order of SLevel of nodes at the beginning but tends to schedule nodes in an ascending order of their TLevel near the end of scheduling process. Fig. 2 (c) shows scheduling of the fig. 1 graph using DLS algorithm on two processor elements.

2.4 The ETF Algorithm

The ETF (Earliest Time First) algorithm [31] computes the earliest start times for all nodes in ready list by investigating the start time of a node on all processors exhaustively. Then, it selects the node that has the smallest start time; ties are broken by scheduling the node with the higher SLevel priority. Scheduling of the fig. 1 graph using EST algorithm on two processor elements is shown by fig. 2 (d).

3. Learning Automata

Learning automata (LA) are, by design, "simple agents for doing simple things," and was first publicized by Narendra and Thathachar [36]. Since then, LA has been used successfully in many applications. The full potential of an LA is realized when multiple automata interact with each other. Interaction may assume different forms such as tree, mesh, array, etc. Depending on the problem that needs to be solved, one of these structures for interaction may be chosen. In most applications, local interaction of LA, which can be defined in a form of graph such as tree, mesh, or array, is more suitable [42].

Variable structure learning automata is represented by a 4-tuple $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{p}, T\}$ where $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$, $\mathbf{p} = \{p_1, p_2, \ldots, p_r\}$, T, r, and m are action-set, inputs-set (environment response), action-probability-vector, the learning algorithm,

Copyright ⓒ 2015 SERSC

number of permitted action of automata, and number of inputs (reactions of environment) respectively. The learning algorithm is a recurrence relation and is used to modify action-probability-vector **p** of the automaton. Various learning algorithms have been reported in the literature. Below, a learning algorithm, called linear learning algorithm is given. Let α_i be the action chosen at stage n as a sample realization from probability distribution $p_i(n)$. In this learning algorithm, if the environment response is favorable, the action-probability-vector will be updated using (reward):

$$p_j(n+1) = \begin{cases} p_j(n) + a \times (1 - p_j(n)) & if \quad j = i \\ (1-a) \times p_j & if \quad j \neq i \end{cases} \tag{7}$$

Moreover, if the environment response is unfavorable, the action-probability-vector will be updated by (penalty):

$$p_j(n+1) = \begin{cases} p_j(n) \times (1-b) & if \quad j = i \\ \frac{b}{r-1} + (1-b) \times p_j(n) & if \quad j \neq i \end{cases} \tag{8}$$

where a and b are *reward* parameter and *penalty* parameter respectively should be tuned experimentally. If $a = b$, the algorithm will be called L_{R-P}, else if $b < a$, the algorithm will be $L_{R-\varepsilon P}$, and else if $b = 0$, the algorithm will be named L_{R-I}.

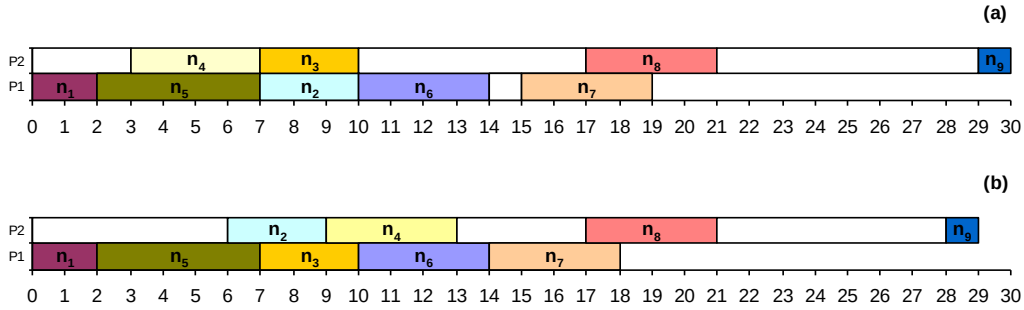
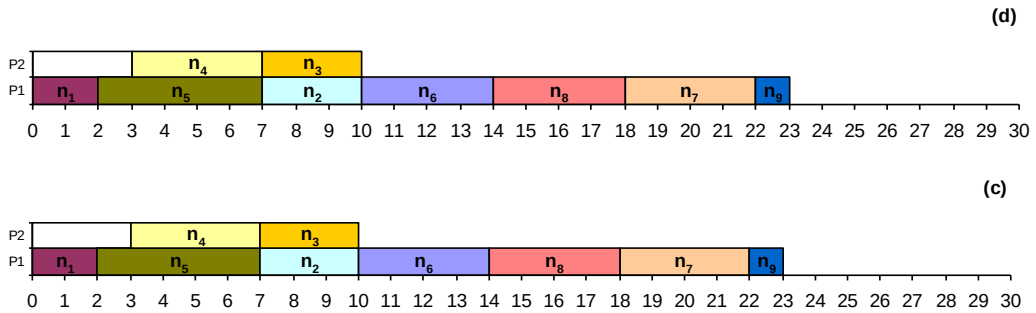


Figure 2. Scheduling of the Fig.1 Task Graph using Four Traditional Heuristics. (a) The HLFET Algorithm. (b) The MCP Algorithm. (c) The DLS Algorithm. (d) The ETF Algorithm.



4. The Proposed Approach

In the proposed approach called LA-MTS, each task in the task graph is represented by a learning automaton. So, a graph of learning automata is generated regarding the given task-graph. Then, an iterative algorithm is executed (fig. 3) so that each iteration like t results in a solution (a complete scheduling). Each iteration consists of n stage, where n is

 Copyright ⓒ 2015 SERSC

the number of tasks in the task graph. In each stage like *k*, learning automatons that do not have any parents or all of their parents have already been executed, are activated to schedule. Among the activated learning automatons, that which has the highest priority is selected. The selected automaton selects a processor to execute based on its action-probability-vector, that is, the permitted action of an automaton which is to select a processor among the available processors (various priority measures such as *TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* can be used to select an automaton among the ready automatons). Finally, after executing all learning automatons, a problem solution or a complete scheduling is obtained for iteration *k*. if obtained solution in iteration *k*, is better than the previous ones (or equal to the best pervious), learning automata will get a *reward* using (7), else they get a *penalty* by (8). The iterations continue until solutions converge to a unique scheduling. A flowchart of the proposed approach is also demonstrated in fig. 3.

Whereas $L_{R-\varepsilon P}$ learning algorithm guarantees converging, it has been used to adapt the learning automata. However, the *reward* and *penalty* parameter should be tuned experimentally. Table II lists six task graphs and their comments considered to evaluate the proposed LA-MTS. The four first graphs compare the LA-MTS with traditional heuristics for multiprocessor task graph scheduling on bounded number of processors (BNP) [35], and two last graphs evaluate it in comparison with the genetic algorithm.

Table 2. Selected Task Graphs for Evaluating the LA-MTS

Graph	Comments	Nodes	Communication Costs
G1	Kwok and Ahmad [13]	9	Variable
G2	Al-Mouhamed [32]	17	Variable
G3	Wu and Gajski [25]	18	60 and 40
G4	Al-Maasarani [45]	16	Variable
G5	Fig. 1 [14]	9	Variable
G6	Hwang *et al.* [14]	18	120 and 80

5. Implementation and Results

The proposed approach was implemented on a Penitum4 (2.6GHz) desktop computer with Microsoft Windows XP (SP2) platform using Microsoft Visual Basic 6.0 programming language. Quit condition of the algorithm was 50 iterations with same makespan, and maximum number of iteration was set to 2000. Table II lists six task graphs and their comments considered to evaluate the proposed approach. The four first graphs compare the proposed approach with the traditional heuristics not only the BNP algorithms but also UNC ones, and then two last graphs that are G5 and G6 evaluate it in comparison with the genetic algorithm. Set of different experiment to select proper *reward* parameter of learning automata, task priority measurement, and to compare with the other traditional heuristics and genetic algorithm has been done.

5.1 Reward Parameter of Learning Automata

At first, set of experiments has been done to detect the appropriate value of *a* parameter (*reward* parameter). Whereas $L_{R-\varepsilon P}$ learning method has been used to adapt the learning automata, and *b* parameter (*penalty* parameter) should be so small, therefore $b = 0.01 \times a$ has been considered in all experiments.

Copyright ⓒ 2015 SERSC

Fig. 4 shows diagram of archived makespan of LA-MTS based on various values of *a* parameter (mean of 10 times of algorithm execution has been used). The above numbers of diagram are mean of number of algorithm iterations, which demonstrate how much the problem space has been explored by learning automata before it converge to a minimum solution. It can be concluded that 0.1 is a qualified value for *a* parameter. A sample of algorithm execution using this value as *reward* parameter has been shown in fig. 5.

 Copyright ⓒ 2015 SERSC

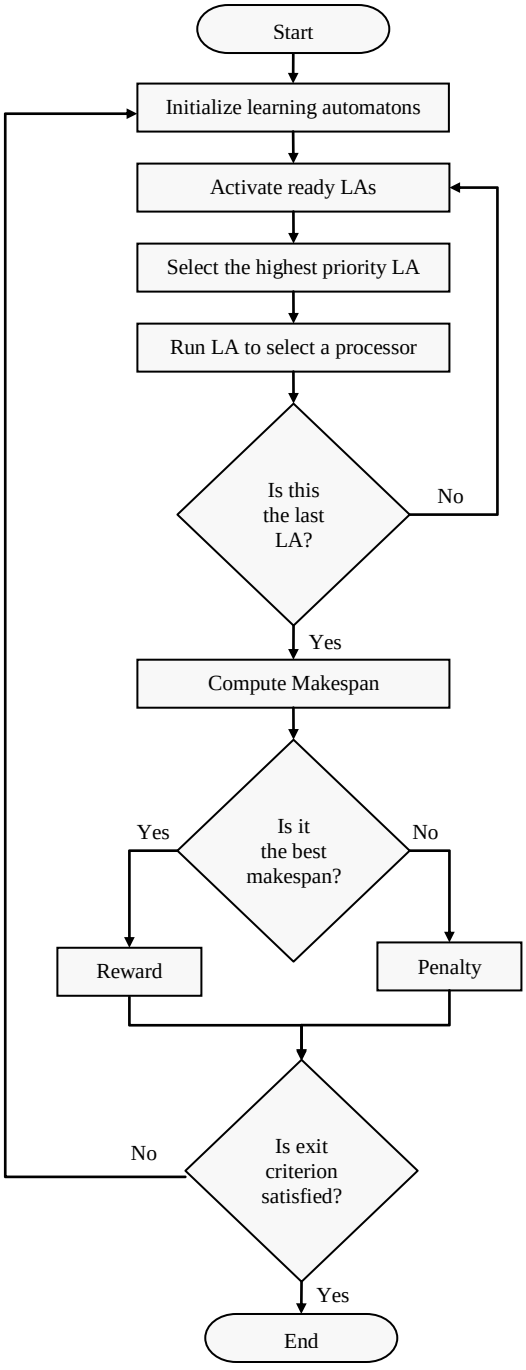


Figure 3. Flowchart of the Proposed Approach

Copyright ⓒ 2015 SERSC

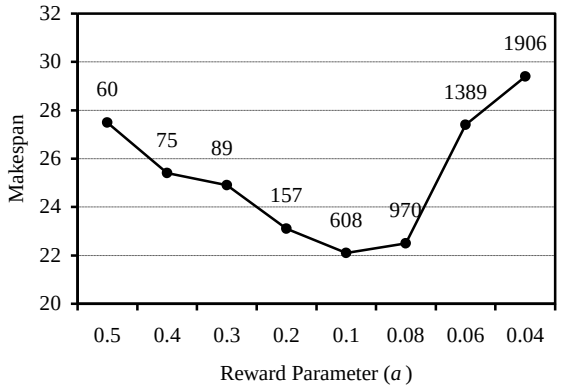


Fig. 4. Diagram of Archived Makespan of LA-MTS Based on Various Values of *a* Parameter (Mean of 10 Times of Algorithm Execution has Been Used). The Above Numbers of Diagram are Mean of Number of Algorithm Iterations

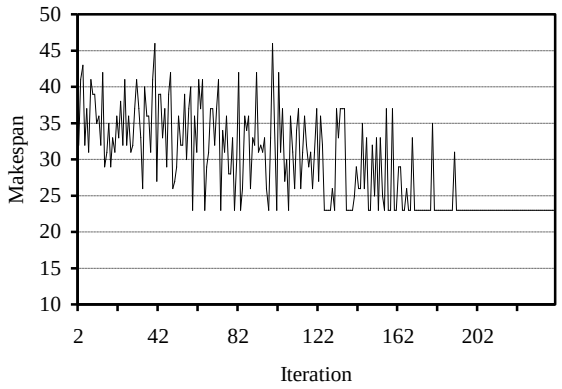


Figure 5. A Sample of Algorithm Execution using a = 0.1. The Algorithm was Terminated after 240 Iteration and the Minimum Makespan was 22 Time-Slot

5.2 Priority Measurement

In primary LA-MTS in each time instance among ready automatons, that which has the least topological number is selected to execute, while various measure of priority can be used intelligently. Therefore, we introduce five new algorithms namely LA-MTST, LA-MTSB, LA-MTSS, LA-MTSA, and LA-MTSN which use *TLevel*, *BLevel*, *SLevel*, *ALAP*, *NOO* as priority measurements of nodes to break ties respectively. Table III and table IV list results of mean and minimum of 10 times of these algorithms execution on the four first graphs. Results of scheduling by LA-MTSN that uses number of offspring of nodes as priority measurements, is very promising; it outperforms the others. LA-MTSN among ready automatons selects one that has the most number of offspring in each time instance. It seems to be better priority measurement. Since, the task, which has lesser offspring or is near the leaves, may be more desirable in the other measurements, which early selecting of it to schedule may increases the overall finish time. Therefore, LA-MTSN will be used in the remained experiments.

 Copyright ⓒ 2015 SERSC

Table 3. Results of Mean of 10 Times of Execution of LA-MTST, LA-MTSB, LA-MTSS, LA-MTSA, AND LA-MTSN

Graph	LA-MTST	LA-MTSB	LA-MTSS	LA-MTSA	LA-MTSN
G1	20	17.4	18.8	17.4	17.1
G2	43.6	44.2	44.8	43.9	43
G3	451	437	428	437	427
G4	58.1	57.3	60.3	58.3	59.4

Table 4. Results of Minimum of 10 Times of Execution of LA-MTST, LA-MTSB, LA-MTSS, LA-MTSA, AND LA-MTSN

Graph	LA-MTST	LA-MTSB	LA-MTSS	LA-MTSA	LA-MTSN
G1	20	17	18	17	17
G2	43	43	43	43	41
G3	440	420	420	410	400
G4	55	56	57	56	58

5.3 Compare with Traditional Heuristics

Whereas the proposed LA-MTS is a BNP algorithm, four BNP heurists namely HLFET, MCP, DLS, and ETF are considered to evaluate it. Fig. 6 shows diagram of mean achieved scheduling of 10 times of algorithm execution and these heuristics for the four first graphs on various number of processors. The diagram bars are normalized to the range of (0, 1] by dividing the archived makespan of each algorithm for each graph by maximum achieved makespan for that graph. It can be seen that LA-MTS has outperformed the heuristics.

5.4 Compare with Genetic Algorithm

The two last graphs evaluate the ACO-MTS compared to one of the best genetic algorithm proposed for multiprocessor task scheduling without task duplication [14]. Table V lists achieved results of not only these two algorithms but also four other traditional ones (BNP as well as UNC methods). Number of processors is large enough to achieve the best possible results for the BNP methods that are, MCP, LA-MTS, and the genetic algorithm. The results show that LA-MTS had the best performance in the first graph and equal to the genetic algorithm in the second one (Note that LA-MTS has used two processors to schedule the first and second

Table 4. The Best Achieved Results of LA-MTN, Genetic Algorithm, and Four Traditional Heuristics [14]

Graph	MCP	DSC	MD	DCP	Genetic	LA-MTS
G5	29	27	32	32	23	22
G6	520	460	460	440	440	440

Copyright ⓒ 2015 SERSC

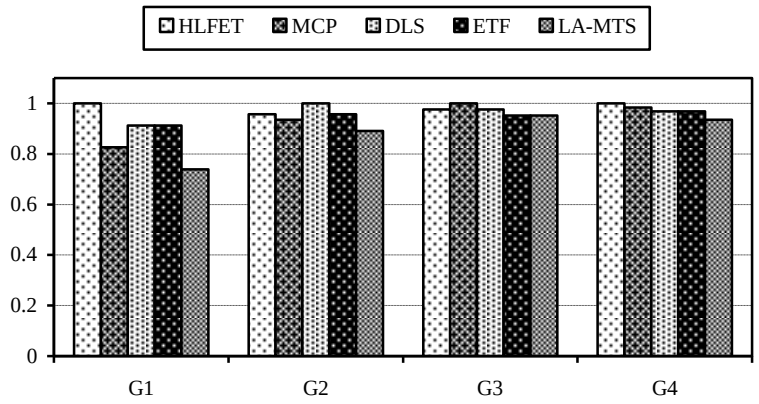


Figure 6. Diagram of Minimum Achieved Scheduling of 10 Times of Algorithm Execution and these Heuristics for the Four First Graphs on Two Processors. The Diagram Bars are Normalized to the Range of (0, 1] by Dividing the Archived Makespan of each Algorithm for Each Graph by Maximum Achieved Makespan for that Graph

graph). Nevertheless, in this genetic algorithm, each generation has 100 chromosomes and maximum number of generations is 1000, that is, it achieves the best scheduling by generating 100,000 solutions while the LA-MTS examines only 2000 complete solutions (2000 iterations) to find the best scheduling. It demonstrates that LA-MTS finds solution so faster than the genetic algorithm.

6. Conclusion

In this paper, a new BNP (Bounded Number of Processors) approach for multiprocessor task-graph scheduling based on a graph of learning automata was introduced. In the proposed approach named LA-MTS, each task in the task graph was represented by a learning automaton. In each stage, the most priority ready automaton was selected to execute on which processor it selects. Among different introduced priority measurements, number of offspring (*NOO*) outperformed the others. The automaton selected one of available processors to schedule in based on its action-probability-vector, until all automatons were scheduled. The learning automata would get *reward* or *penalty* with respect to desirability of the achieved makespan because of generating better scheduling in the next iterations. Six task graphs, which results of the other BNP methods were already specified on them, were elected to evaluate the proposed LA-MTS (scheduling of them using LA-MTS has been shown in fig. 7). The four first graphs evaluated LA-MTS against the traditional BNP heuristics. In a set of experiments, LA-MTS outperformed them. Another set of experiments evaluated LA-MTS in comparison with one of the best genetic algorithm in this field (without task duplication). In one graph, LA-MTS had better performance and in the second graph was equal to GA. While, the genetic algorithm examined too more scheduling to achieve the best result (almost 100,000 scheduling). All of these demonstrate that the proposed LA-MTS is so successful in multiprocessor task scheduling.

References

[1]. S. Parsa, S. H. Lotfi and N. Lotfi, "The approach based on evolutional processing for task graph scheduling in multiprocessor architecture", Proc. 11th Iranian Int. CSI Computer Conf., Tehran, **(2006)**.

 Copyright ⓒ 2015 SERSC

[2]. M. Salmani, M. Zali and M. Moghimi, "Task Scheduling in Multi-Processor Systems Using Genetic Algorithm and Reinforcement Learning", Proc. 12th Iranian Int. CSI Computer Conf., Tehran, **(2007)**, pp. 1948-1951.
[3]. M. Abdeyazdan and A. Rahmani, "Task scheduling in multiprocessor systems using a new genetic algorithm priority based on number of offspring", Proc. 13th Iranian Int. CSI Computer Conf., Kish, **(2008)**.
[4]. E. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Trans. Parallel Distrib. Syst., vol. 5, no. 2, **(1994)**, pp. 113-120.
[5]. M. Salmani, S. Fakhraie, F. Montazeri, S. M. Fakhraie and M. Nili, "A Representation for Genetic-Algorithm-Based Multiprocessor Task Scheduling", IEEE Congr. On Evolutionary Computation, Vancouver, **(2006)**, pp. 340-347.
[6]. R. Correa, A. Ferreira and P. Rebreyend, "Scheduling Multiprocessor Tasks with Genetic Algorithms", IEEE Trans. Parallel Distrib. Syst., vol. 10, no. 8, **(1999)**, pp. 825-837.
[7]. C. W. Zomaya and B. Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", IEEE Trans. Parallel Distrib. Syst., vol. 10, no. 8, **(1999)**, pp. 795-812.
[8]. M. Dehodhi, I. Ahmad and I. Ahmad, "A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms".
[9]. Wu, H. Yu, S. Jin, K. Lin and G. Schiavone, "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling", IEEE Trans. Parallel Distrib. Syst., vol. 15, no. 9, **(2004)**, pp. 824-834.
[10]. E. Hou, R. Hong and N. Ansari, "Efficient Multiprocessor scheduling Based on Genetic Algorithms".
[11]. P. Chretienne, "Scheduling Theory and Its Application", New York: Wiley, **(1995)**.
[12]. Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", Hong Kong Research Grants Council, Hong Kong, Rep. HKUST 734/96E and HKUST 6076/97E, **(1998)**.
[13]. Ahmad and Y. Kwok, "On Parallelizing the Multiprocessor Scheduling Problem", IEEE Trans. Parallel Distrib. Syst., vol. 10, no. 8, **(1999)**, pp. 795-812.
[14]. R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs", Computer & Operations Research, vol. 35, **(2008)**, pp. 976-993.
[15]. B. Kruatrachue and T. G. Lewis, "Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems", Oregon State University, Corvallis, Tech. Rep. OR 97331, **(1987)**.
[16]. C. H. Papadimitriou and M. Yannakakis, "Scheduling Interval-Ordered Tasks", SIAM J. Computing, vol. 8, **(1979)**, pp. 405-409.
[17]. J. Y. Colin and P. Chretienne, "C.P.M. Scheduling with Small Computation Delays and Task Duplication", Operations Research, **(1991)**, pp. 680-684.
[18]. Y. C. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors", Proc. Supercomputing, **(1992)**.
[19]. H. Chen, B. Shirazi and J. Marquis, "Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication", Proc. Int'l Conf. Parallel and Distributed Systems, **(1993)**.
[20]. Ahmad and Y. K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling", IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 9, **(1998)**, pp. 872-892.
[21]. M. A. Palis, J. C. Liou and D.S.L. Wei, "Task Clustering and Scheduling for Distributed Memory Parallel Architectures", IEEE Trans. Parallel and Distributed Systems, vol. 7, no. 1, **(1996)**, pp. 46-55.
[22]. G. L. Park, B. Shirazi, and J. Marquis, "DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems", Proc. 11th Int'l Parallel Processing Symposium, **(1997)**.
[23]. S. J. Kim and J. C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures", Proc. Int'l Conference on Parallel Processing, **(1988)**.
[24]. V. Sarkar, "Partitioning and Scheduling Parallel Programs for Multiprocessors, Cambridge, MA: MIT Press, **(1989)**.
[25]. M. Y. Wu and D. D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems", IEEE Trans. Parallel and Distributed Systems, vol. 1, no. 3, **(1990)**, pp. 330-343.
[26]. T. Yang and A. Gerasoulis, "List Scheduling with and without Communication Delays", Parallel Computing, vol. 19, **(1993)**, pp. 1321-1344.
[27]. Y. K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", IEEE Trans. Parallel and Distributed Systems, vol. 7, no. 5, **(1996)**, pp. 506-521.
[28]. T. L. Adam, K. M. Chandy and J. Dickson, "A Comparison of List Scheduling for Parallel Processing Systems", Comm. ACM, vol. 17, no. 12, **(1974)**, pp. 685-690.
[29]. C. McCreary and H. Gill, "Automatic Determination of Grain Size for Efficient Parallel Processing", Comm. ACM, vol. 32, **(1989)**, pp. 1073-1078.
[30]. Baxter and J. H. Patel, "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm", Proc. 1989 Int'l Conf. Parallel Processing, **(1989)**.
[31]. J. J. Hwang, Y. C. Chow, F. D. Anger and C. Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times", SIAM J. Computing, vol. 18, no. 2, **(1989)**, pp. 244-257.

Copyright ⓒ 2015 SERSC

[32]. M. A. Al-Mouhamed, "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs", IEEE Trans. Software Engineering, vol. 16, no. 12, **(1990)**, pp. 1390-1401.
[33]. P. Shroff, D. Watson, N. Flann and R. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments", 5th IEEE Heterogeneous Computing Workshop (HCW), **(1997)**, pp. 98-104.
[34]. M. Wu, W. Shu and J. Gu, "Local Search for DAG Scheduling and Task Assignment", Proc. Int'l Conf. Parallel Processing, **(1997)**.
[35]. Y. Kwok and I. Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms", Hong Kong Research Grants Council, Hong Kong, Rep. HKUST 734/96E and HKUST 6076/97E, **(1999)**.
[36]. S. Narendra and M. A. L. Thathachar, "Learning Automata: A Survey", IEEE Trans. Syst., Man, Cybern., vol. SMC-14, **(1974)**, pp. 323–334.
[37]. G. I. Papadimitriou, M. S. Obaidat and A. S. Pomportsis, "On the use of learning automata in the control of broadcast networks: A methodology", IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 32, no. 6, **(2002)**, pp. 781–790.
[38]. S. Misra, K. I. Abraham, M. S. Obaidat and P. V. Krishna, "LAID: A learning automata-based scheme for intrusion detection in wireless sensor networks", Security Commun. Netw., vol. 2, no. 2, **(2009)**, pp. 105–115.
[39]. E. Fayyoumi and B. J. Oommen, "Achieving micro aggregation for secure statistical databases using fixed structure partitioning based learning automata", IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 9, no. 5, **(2009)**, pp. 1192–1205.
[40]. H. Beigy and M. R. Meybodi, "Utilizing distributed learning automata to solve stochastic shortest path problems", Int. J. Uncertain., Fuzziness Knowl.-Based Syst., vol. 14, no. 5, **(2006)**, pp. 591–615.
[41]. S. Misra and B. J. Oommen, "Using pursuit automata for estimating stable shortest paths in stochastic network environments", Int. J. Commun. Syst., vol. 22, no. 4, **(2009)**, pp. 441–468.
[42]. H. Beigy and M. R. Meybodi, "Cellular Learning Automata With Multiple Learning Automata in Each Cell and Its Applications", IEEE Trans. Syst., vol. 40, no. 1, **(2010)**, pp. 54–65.
[43]. L. Thathachar and P. S. Sastry, "Varieties of Learning Automata: An Overview", IEEE Trans. Syst., vol. 32, no. 6, **(2002)**, pp. 711–722.
[44]. H. R. Boveiri, "ACO-MTS: A New Approach for Multiprocessor Task Scheduling Based on Ant Colony Optimization", 3rd Int. Conf. on Intelligent and Advanced Systems.
[45]. Al-Maasarani, "Priority-Based Scheduling and Evaluation of Precedence Graphs with Communication Times", M.S. Thesis, King Fahd University of Petroleum and Minerals, **(1993)**.
[46]. G. C. Sih and E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 2, **(1993)**, pp. 75-87.

Author

Hamid Reza Boveiri received the B.Sc. degree from Birjand University, Birjand, Iran, in 2005, and M.Sc. degree from Islamic Azad University Science and Research Branch, Ahwaz, Iran, in 2009, both in software engineering.

He is currently a faculty member of computer department of Sama College, Islamic Azad University, Shushtar Branch, Shushtar, Iran. He is also member of Young Researchers Club of Islamic Azad University, Shushtar Branch, and there, he is the head advisor of research workgroup. His research interests include optimization, meta-heuristics, signal processing, and pattern recognition.

 Copyright ⓒ 2015 SERSC