LA-PR: Priority Task Graph Scheduling Based on Learning Automata for Real Time Systems

(IJCSNT) International Journal of Communication Systems and Network Technologies
ISSN Number: 2053-6283
Volume 1, Issue No. 1, 41–53

https://journals.mirlabs.in/index.php/ijcsnt
10.18486/ijcsnt/1.1.004

Yosef Masoudi[1], Shahriar Lotfi[2], Davod Karimzadgan[3]

Abstract

One of the main issue and main challenge in real time systems is achieving to acceptable results in acceptable time. a job in multiprocessor real time systems divide in the set of the tasks with the relation between them in order that one task can be execute only when it's parents executed. by difficulty in scheduling the task graph and it's complexity, many effort accomplish for finding the best optimized solution. In this paper, we tried to have the balance between the processor and also reduce relation between the processor and importance of them we tried to improved the speed of getting the response. In most of the activity and experiment, the run time of the scheduling algorithm ignored. finally, the result of maintaining this solution show that we can have the appropriate schedule in acceptable time. also in this paper, at the end, we compared the proposal algorithm with the other famous scheduling algorithm.

Keywords

Multiprocessor Scheduling, Real Time Systems, Task Graph, Learning Automata

Received: 29 December 2011; **Revised:** 31 March 2012; **Accepted:** 09 April 2012;
Published: 18 April 2012;

[1]Al Zahra Junior college of Tabriz, East Azerbaijan Province, Iran.
[2]Computer Science Department, University of Tabriz, East Azerbaijan Province, Iran.
[3]Payam Noor University of Tehran, Tehran Province, Iran.

Corresponding author:
Email: d_karimzadgan@pnu.ac.ir

I. Introduction

In real time systems we have the limited time that each job must be execute in this time that is if one job can't be ended in it's limited time may be have the disastrous result. accordingly the having the appropriate scheduling algorithm in real time systems is essentially necessary. main purpose in multiprocessor scheduling is running a parallel program on multiple processor so if running time of all program in order of running time of jobs and relation between them be minimized. also time of running the scheduling algorithm is one of the important problem that could not be ignored. number of processor is a input parameter for scheduling algorithm which if we plan the good scheduling it will be have the desirable speed for program.

the ways of scheduling is dividing in two group: homogeneous and heterogeneous. In homogeneous type, power of processing the all processes is equal but in heterogeneous type, we have the different power between the processes. for increasing the speed of the result in multiprocessor real time systems, a job dividing into the set of tasks (task graph) and in dividing the job, we must preserve the prerequistive relation between them. Each task have the running time that it can be execute only when it's parent be executed. Also we have the directed edge from the parent task to the child task which it's number show the cost of relation between them and if both of the parent and child tasks executed in the same processor the relational cost will be zero.

In this paper we schedule the processor by the learning automata also we allocate one learning automata for each processor rather than tasks. The automata in order to selected operation, rewarded or be penalized.

Scheduling in real time systems with imprecise computation studied by Georgios L. Stavrinides and Helen D. Karatza[1]. Scheduling directed acyclic graph by combination of genetic algorithm and learning automata in heterogenous systems studied by habib izadkhah[2]. Scheduling in distributed real-time systems has been studied by many authors[7–14].

II. The Problem

in homogenous distributed real time system there is queues that each queue assigned to a processor. When each job such as Jk arrive, it containing a graph without direct cycle of the components of exclusive tasks. Each graph node represents task Ti of job Jk and a direct edge from vertex Ti to Tj indicates that a message should be transmit from vertex Ti to Tj. A task can run when it gets all necessary data from it's parents. A task having no priority is called input task and a task being the last one is called output task. The first prior of a task is called the parent of the task and the first task following a task is called the child. A relation cost is assigned to each task edge. Each pair of processors may have different relation costs and the relation cost of different tasks in a processor is ignored. Also it is supposed that different processors inside a system are strongly connected to each other and even they can send or receive data during task run time. If both parent and child tasks run in the same processor, the relation cost would be zero. When a job comes in turn, it may have several input and output tasks. Task Ti of a job Jk includes the following features[1]:

1. service time= Ci
2. a set of parents of a task= Vi
3. a set of childs of a task= Yi

4. the relation cost of a task Ti to task Tj= Cij
5. start time of task i= Si
6. end time of task i= Ei
7. the processor assigned to task i = Pi

In order to produce random directional graph without cycling used from following algorithm[1].

```
Algorithm random DAG generator
Input: the maximum number of tasks in the DAG
Output: A DAG in the from of a topological order list y of its tasks
1:  determine the number of tasks in the graph N~U(1,MAX)
2:  if N>1 then
3:    determine the number of entry tasks in the graph Nentry N~U(1,N-1)
4:    create the first entry task T1
5:    set R1 =1
6:    place task T1 in the topological order list y
7:    repeat
8:      create a task T1
9:      determine the number of Tj's parent tasks |vj|~ U(1,|y|)
10:     repeat
11:       choose randomly a task Ti in y
12:       create edge Eij that connects tasks Ti and Tj
13:     until |Vj| parent tasks are connected to the Tj edge.
14:     set Rj=max TiEVj {Ri}+1
15:     place task Tj in the topological order list y according to its rank Rj
16:   until N-Nentry non-entry tasks are created
17:   repeat
18:     create a task Tm
19:     determine the number of Tm's child tasks |ym|~U[1,N-Nentry]
20:     repeat
21:       choose randomly a non-entry task Tn in y
22:       create edge Emn that connects tasks Tm and Tn
23:     until |ym| child tasks are connected to task Tm
24:     set Rm=1
25:     place task Tm in the topological order list y according to its rank Rm
26:   until Nentry -1 entry tasks are created
27:   else
28:     create the only task T in the graph
29:   end if
```

Figure (1) is a sample task graph that the algorithm has produced. The number inside a node shows runtime and the ones on the edges states relation cost.

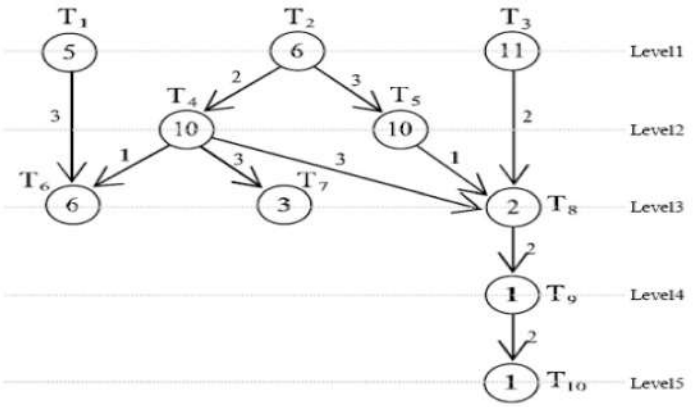

Figure 1. sample of DAG[1]

III. Learning Automata

A learning automata is an imaginary model that selects an operation from its finite set of operations by random and exerts it over the circumference. The circumference evaluates the selected operation by learning automata and announces the result by means of an amplifying signal. learning automata updates the amplifying signal of its interior state by the selected operation and then selects the next operation. circumference can be shown by the triple E={a, b, c} in which a= $\{a_1, a_2, ...\}$ is the set of input data and B= $\{b_1, b_2, b_3, ...\}$, the set of output data and c= $\{c_1, c_2, c_3, ...\}$ is the set of penalize probability. when set B has two elements, the circumference will be the same as p. In such circumference B_1=1 is considered as a fine and B_2=0 as a reward. In a circumference of Q set B includes the finite number of elements. And in a circumference of S, set B includes the infinite number of elements. Ci is fining probability of operation ai. learning automats are divided into two groups with variable structures[3–7].

IV. Proposed Algorithm

In proposed method a learning automata is assigned to each processor rather than assigning to jobs and it is rewarded or penalized based on the selected operation. If selected operation be proper operation then it rewarded else penalized. In other implement by learning automata, the automata allocated to task but in this paper we allocate them to processor that this issue lead to we have least computation relative other methods. The purpose of scheduling is to assign the tasks to the processors in a way that work balance among the processors is maintained and the relation cost decreases and also scheduling is done in an ideal time. So the result is acceptable for the system. All tasks are laid in one queue. Each processor has it's own queue and when the processor want to execute a job, referral to it's queue[1].

Every automata has a C_{time} that represent processor current time. At the beginning C_{time} value is set to zero for all automata. Si value of a task selected for scheduling equals to C_{time} of the selected processor. Each automata having smaller C_{time}, selects an operation by random and this is rewarded

or penalized. The mentioned action includes selecting task from a queue which tasks are laid in. An automata having one of the conditions below is rewarded: If the selected task is an input task or If the parents of the selected task have executed and relation cost is also considered in the other words:

$$\text{For all } j \text{ and } V_i \quad (C_{time} < C_j + C_{ij} + S_j) \tag{1}$$

If tasks i, j are both run on the same processor then C_{ji} value will be zero. In the other words if an automata has selected task i then parent of task i (task j) should be in the automata local list and is added to its own local list and the values Si for ith task and C_{time} for Kth automata are updated as follows:

$$S[i] = C_{time}[k] \tag{2}$$

$$C_{time}[k] = C_{time}[k] + C[i] \tag{3}$$

If an automata such as K in a predefined range can not be rewarded it will be penalized and its Ctime is updated as follows.

$$C_{time}[k] = \max(C_{time}) + 1 \tag{4}$$

The algorithm continues until all tasks are scheduled. Following algorithm indicates the pseud code of the proposed algorithm.

```
Algorithm allocate learning automata for each processor
```

```
1:  Procedure scheduling
2:  input: directed acyclic graph(DAG)
3:  while all task that are in queue not scheduled
4:    the processor that have the minimum time select a task from queue
5:    if parent of the selected task are executed and the result from parent ar
6:    to his child task then
7:      the automata rewarded and update cpu time
8:    Else
9:      Penalized automata with
       Ctime[k] = max(Ctime) + 1
10:   End if
11: End while
12: End of procedure
```

V. Evaluation and Experimental Results

The following table (table(1)) represent and describe the algorithms that compared with proposed algorithm. The canonical that used to compare algorithms are: time of execution of scheduling algorithm, time of result production, actual time that is combination of time of execution of scheduling algorithm and time of result production, stability of execution time of scheduling algorithm and stability of result

production. These canonicals are very important in real time systems in order to achieving to acceptable result.

The two example of directed acyclic graph represented in figure(3) and figure(4).

Table 1. compared algorithms and their description

Algorithm name	Description
LA-HLF	This algorithm implemented by learning automata and operate based on highest level first if more than one task ready to scheduling, the task that have minimum task number selected to Scheduling. in this algorithm, automata allocated to task.
LA-LSTF	This algorithm implemented by learning automata and operate based on least space time first. if more than one task ready to scheduling, the task that have minimum task number selected to scheduling. In this algorithm, automata allocated to task.
LA-GA	This algorithm is combination of genetic algorithm and learning automata. the genetic operators applied randomly. In this algorithm used of complex state of learning automata with 6% reward and penalized rate.

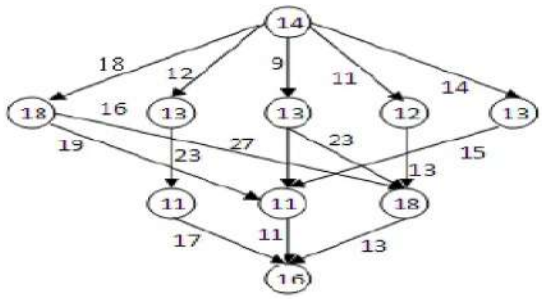

Figure 2. directed acyclic graph[2]

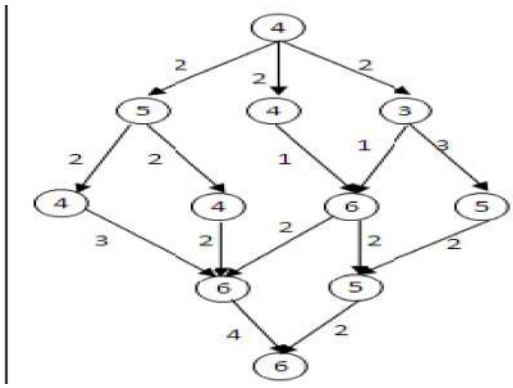

Figure 3. another example of DAG[2]

Consider the graph of figure(1) as graph1 and graph of figure(3) as graph2 and finally graph of figure(2) as graph3. LA-PR(proposed algorithm), LA-GA, LA-LSTF, LA-HLF algorithms applied to graph of figure(1) and regardless of algorithm execution time, result of table(2) generated. Protducted result s are based on second.

Table 2. make spam of three graph

Algorithm name	Graph1	Graph2	Graph3
LA-HLF	23	32	97
LA-LSTF	23	31	102
LA-GA	22	30	98
LA-PR	22	31	98

According to table(2), for graph1 LA-GA and LA-PR are better than LA-HLF and LA-LSTF. For graph2, LA-LSTF and LA-PR are better than LA-HLF but also LA-GA is better than LA-PR and LA-LSTF. For graph3, LA-HLF is better than all. LA-GA and LA-PR are better than LA-LSTF.

The time of execution of proposed algorithm and compared algorithm assembeled in table(3). Producted results are based on ms.

Table 3. time of execution on three graph

Algorithm name	Graph1	Graph2	Graph3
LA-HLF	270	311	510
LA-LSTF	270	311	510
LA-GA	1573	1672	1982
LA-PR	45	16	98

In table(3), proposed algorithm from point of time of execution of algorithm is better than all algorithm and have many difference from other. GA-LA is worst than all and in average state it's termination time for three graph is 1742 ms. LA-HLF and LA-LSTF have similar time of execution but better than LA-GA

and worst than of LA-PR. In the table(4) actual time completion that is combination of time of execution of scheduling algorithm and time of execution of job, for three graph are presented.

Table 4. actual termination time

Algorithm name	Graph1	Graph2	Graph3
LA-HLF	23.27	32.31	97.51
LA-LSTF	23.27	31.31	102.51
LA-GA	23.57	31.67	99.98
LA-PR	22.04	31.01	98.38

Looking in table(4), reveal that in overall state proposed algorithm are better than another. These resulat obtined in average state and them shows proposed algorithm in overall state is better than all. For meet stability of time of execution of scheduling algorithm, stability of result production and stability of actual termination time looking to figure(4) through figure(13).

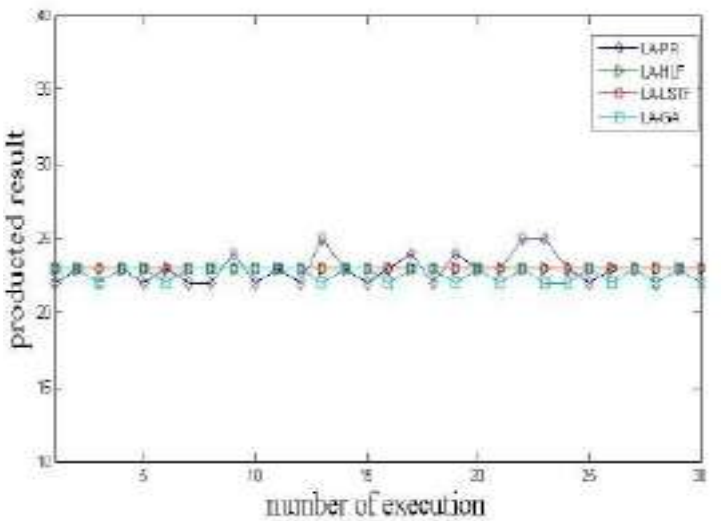

Figure 4. stability of algorithms on graph1 based on result production

In figure(4) through figure(6) the stability of algorithms based on result production on three graph are represented in 30 continuous execution. In these figures the LA-LSTF and LAHLF absolutely are stable but LA-PR and LA-GA have fluctuation. these fluctuation are not much. In overall state, LA-PR and LA-GA are better than LA-HLF and LA-LSTF. In figure(7) through figure(9) the stability of algorithms based on time of execution of scheduling algorithm on three graph are represented. LA-PR is always better than another. LA- HLF and LA-LSTF are similar. LA-GA due to complex computation have many difference with another also is worst than another. In figure(10) through figure(12), the stability of algorithms based on actual termination time are represented.

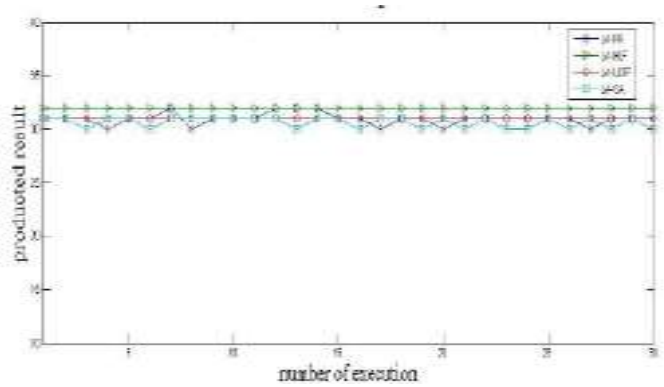

Figure 5. stability of algorithms on graph 2 based on result production

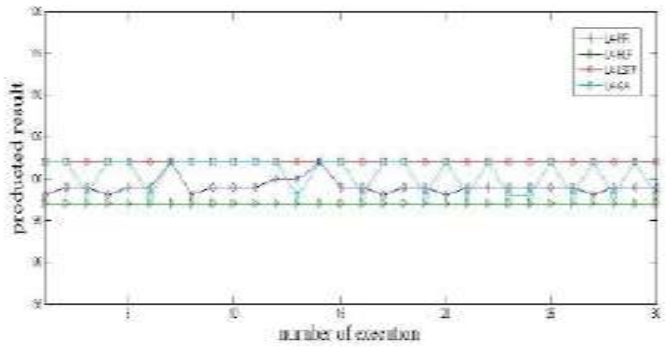

Figure 6. stability of algorithms on graph 3 based on result production

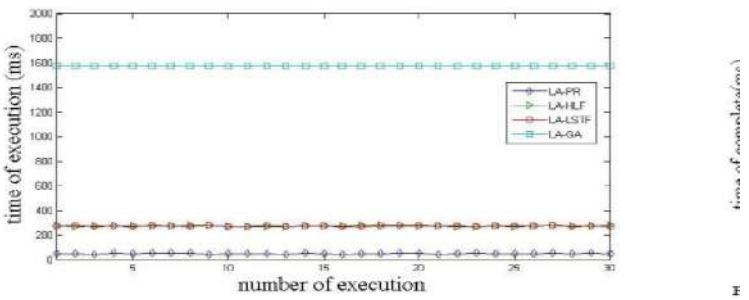

Figure 7. stability of algorithms on graph1 based on time of execution of scheduling algorithm

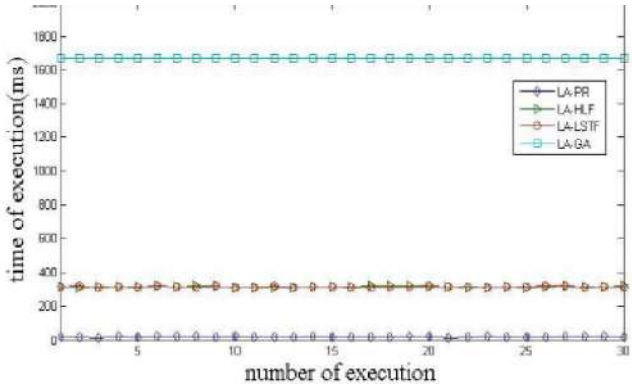

Figure 8. stability of algorithms on graph 2 based on time of execution of scheduling algorithm

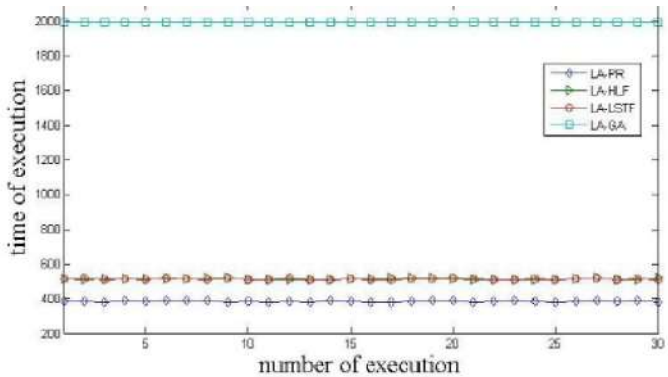

Figure 9. stability of algorithms on graph 3 based on time of execution of scheduling algorithm

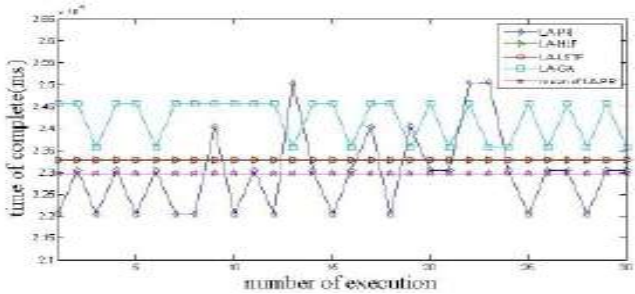

Figure 10. stability of algorithms on graph1 based on actual termination time

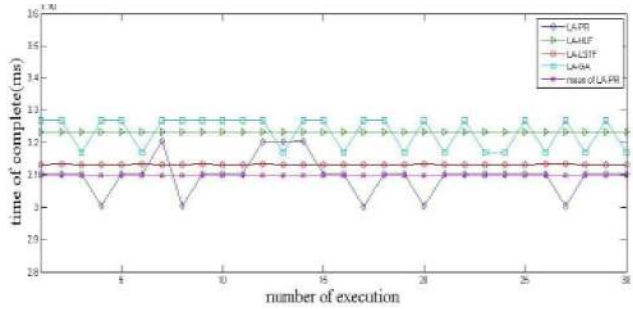

Figure 11. stability of algorithms on graph 2 based on actual termination time

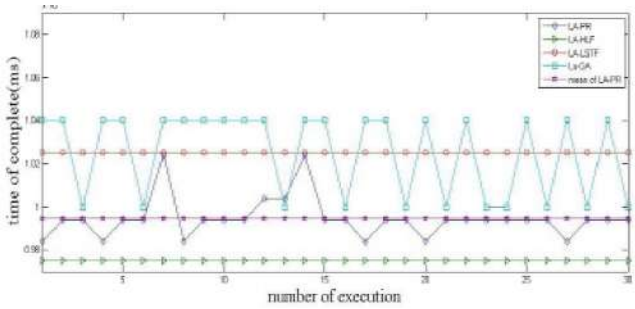

Figure 12. stability of algorithms on graph 3 based on actual termination time

LA-LSTF and LA-HLF approximately are stable but LA-PR and LA-GA have small fluctuation. For graph1, the LA-HLF and LA-LSTF are overlap. For graph2 LA-LSTF is better than LA-HLF and for graph3 LA-HLF is better than LALSTF. LA-PR for graph1 and graph2 is better than LA-HLF but in graph3 is worst. Also in figure(10) through figure(13) diagram of average state of LA-PR is drawn. LA- GA has worst performance relative to another algorithm. If we consider that jobs in real time system arrival with combination of graph1, graph2 and graph3, the figure(13) represent the stability of algorithms based on actual termination time. Observe that proposed algorithm(LA-PR) in overall state have best

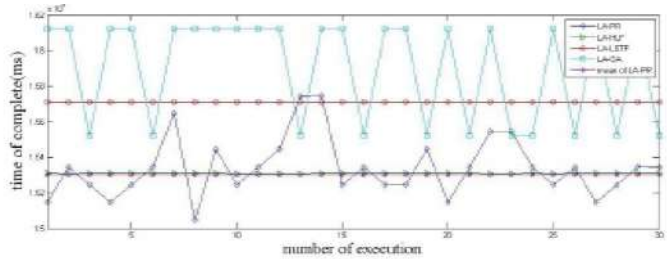

Figure 13. overall stability of algorithms on all graph based on actual termination time

performance, then LA-HLF is better than LA-LSTF and LA-GA and finally the LA-GA has worst functionality.

VI. Conclusion and Future Work

In this paper, we evaluated by simulation the performance of the LA-PR, LA- HLF, LA-LSTF and LA-GA policies for the scheduling of multiple task graphs in a homogeneous multi processor real-time system. For each algorithm, in this paper presented new versions that implemented by learning automata. Our simulation results show that LA-PR in overall state is better than other mentioned algorithms. However, in this paper we did not take into account the case where the system is heterogeneous. Furthermore, our scheduling policies could be further enhanced, by considering and exploiting schedule holes that may appear due to the data dependencies of the tasks. Therefore, further research is needed in order to address these issues.

References

[1] Georgios L. Stavrinides, Helen D. Karatza, scince direct, “Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations”, 2010.

[2] izadkhah h, lotfi sh, isazadeh a, “task graph scheduling in heterogeneous systems”, 16th international conference of iran, 2010.

[3] K. S. Narendra and M. A. L. Thathachar, “Learning Automata: An introduction”, Prentice Hall, 1989.

[4] Meybodi, M. R. and Beigy, H., “New Class of Learning Automata Based Scheme for Adaptation of Back-propagation Algorithm Parameters”, Proc. Of EUFIT-98, Sep. 7-10, Achen, Germany, pp. 339-344, 1998.

[5] Oommen, B. J. and Ma, D. C. Y., “Deterministic Learning Automata Solution to the Keyboard Optimization Problem”, IEEE Trans. On Computers, Vol. 37, No. 1, pp. 2-3, 1988.

[6] Beigy, H. and Meybodi, M. R., “Optimization of Topology of neural Networks Using Learning Automata”, Proc. Of 3th Annual Int. Computer Society of Iran Computer Conf. CSICC-98, Tehran, Iran, pp. 417-428, 1999.

[7] Wenming Li, Krishna Kavi, Robert Akl, scince direct, “A nonpreemptive scheduling algorithm for soft real-time systems.” 2007.

[8] Adam, T.L., Chandy, K.M., Dickson, J.R., “A comparison of list schedules for parallel processing systems. Communications of the ACM 17”, 685-690, 1974.

[9] Buttazzo, G.C., “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, second ed. Springer-Verlag”, New York., 2004.

[10] Chen, H., Maheswaran, “Distributed dynamic scheduling of composite tasks on grid computing system’s. In: Proceedings of the 2002 International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, FL, April 2002.

[11] Cheng, B.C., Stoyenko, A.D., Marlowe, T.J., Baruah, S.K, “LSTF: a new scheduling policy for complex real-time tasks in multiple processor systems. Automatica 33 (5)”, 921-926, 1997.

[12] Feng, W.C., “Applications and extensions of the imprecise computation model”, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1996.

[13] Feng, W.C., Liu, J.W.S., “Algorithms for scheduling real-time tasks with input error and end-to-end deadlines”. IEEE Transactions on Software Engineering 23, 93-106, 1997.

[14] Han, C.C., Shin, K.G., Wu, J., “A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults.” IEEE Transactions on Computers 52, 2003.