A novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by using Elitism stepping method

AMIR MASOUD RAHMANI[1], MOHAMMAD ALI VAHEDI[2]

[1] Computer Engineering Department, Islamic Azad University, Science and Research branch, Tehran, Iran.
[2] Payame nour university, Iran.
[1] rahmani@sr.iau.ac.ir
[2] pnu_sis@gmail.com

Abstract. Task scheduling is essential for the suitable operation of multiprocessor systems. The aim of task scheduling is to determine an assignment of tasks to processors for shortening the length of schedules. The problem of task scheduling on multiprocessor systems is known to be NP-complete in general. Solving this problem using by conventional techniques needs reasonable amounts of time. Therefore, many heuristic techniques were introduced for solving it. This paper presents a new heuristic algorithm for task scheduling, based on evolutionary method which embeds a new fast technique named Elitism Stepping into Genetic Algorithm (GA). By comparing the proposed algorithm with an existing GA-based algorithm, it is found that the computation time of the new algorithm to find a sub-optimal schedule is decreased; however, the length of schedule or the finish time is decreased too.

Keywords: Task scheduling, Multiprocessor Systems, Genetic Algorithm, Elitism Stepping.

(Received December 10, 2007 / Accepted May 26, 2008)

1 Introduction

The problem of scheduling parallel tasks onto multiprocessors is to simply allocate a set of tasks to processors such that the optimal usage of processors and accepted computation time for scheduling algorithm are obtained. The assumption of this paper is based on the deterministic model, that is, the number of processors, the execution time of tasks, the relationship among tasks and precedence constraints are known beforehand. The precedence constraints between tasks are represented by a Directed Acyclic Graph (DAG). In addition, the communication cost between two tasks is negligible and the multiprocessor system is uniform and non-preemptive, that is, the processors are identical, and each processor completes the current task before the new one starts its execution.

The complexity of the scheduling problem is very depended to the DAG, the number of processors, the execution time of tasks and also the performance criteria which would to be optimized. To date, many heuristic methods have been presented to schedule tasks on multiprocessor systems [1, 2, 3, 4, 6, 10]. Also, there are many studies have been used for task scheduling based on GAs [5, 8, 9, 11, 12, 13, 14]. GAs are a problem solving strategy, based on Darwinian evolution, which has been successfully used for optimization problems [7].

The aim of this paper is to present a GA which uses a novel proposed method, named Elitism Stepping to decrease the computation time for finding a sub-optimal schedule. However, this new method is general and could be applied to any GA.

The remainder of this paper is organized as follows: The task graph model or DAG is described in section 2. The previous scheduling algorithm named, Basic Genetic Algorithm (BGA) is explained in section 3. In section 4, the proposed scheduling algorithm is described. The results of the simulation studies are presented in section 5. The paper concludes with section 6.

2 Task Graph Model

A set of tasks could be represented by a DAG. The task graph $G=(V,E)$ is a DAG which has a set of nodes V and a set of directed edges E which connect the nodes to each other. The V is a set of m tasks to be executed, so $V=\{T_1,T_2,\dots T_m\}$. The directed edges are represented by $E=\{e_{ij}\}$ which e_{ij} is an edge between two tasks T_i and T_j specifies that T_i must be completed before T_j can start; the notation of $T_i \geq T_j$ is used for this purpose and T_i is a predecessor of T_j and T_j is a successor of T_i.

If there is a path of the directed edges from T_i to T_j then T_i is an ancestor of T_j and T_j is a successor of T_i.

A set of predecessors of task T_i is denoted by $PRED(T_i)$.

A set of successors of task T_i is denoted by $SUCC(T_i)$. The height of a task is defined as Equation 1 [8]:

$$height(T_i) = \begin{cases} 0 & If\ PRED(T_i) = \phi \\ & otherwise \\ 1 + \max\ height(T_j) \mid T_j \in PRED(T_i) & \end{cases} \quad (1)$$

By definition, for any two tasks T_i and T_j, if T_i is an ancestor of T_j, then T_i has larger value of height than T_j. If there is no relationship between two tasks, then they could execute in any arbitrary order. A DAG which has eight tasks according to their height and their execution time (the time needed for a task to execute) is shown in Figure 1.

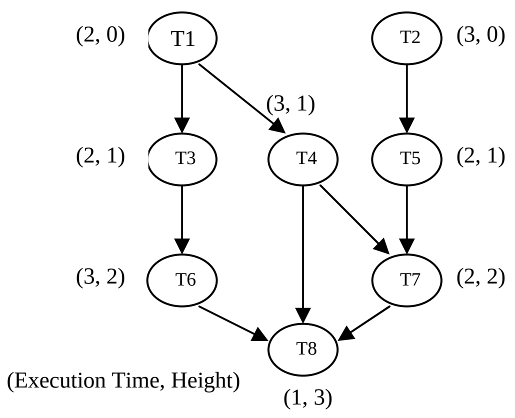

Figure 1- An example of a DAG.

Figure 2 shows a legal schedule on two processors for a DAG as shown in Figure 1, which precedence constraints are considered. The length of schedule or the total finish time is 11.

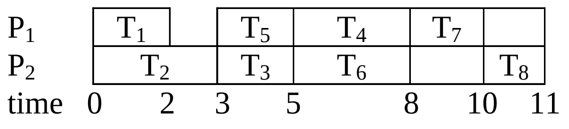

Figure 2- A legal schedule for a DAG of Figure 1

3 Basic Genetic Algorithm (BGA)

The genetic algorithms GAs have been widely and successfully used for many optimization problems [7]. GAs are probabilistic techniques that start from an initial population of generated potential solutions to a problem, and regularly evolve towards better solutions through a repetitive application of genetic operators such as crossover, mutation, selection and reproduction [8].
In this paper, the presented algorithm by Hou et al. [8] in multiprocessor task scheduling is chosen as a Basic Genetic Algorithm (BGA) which is discussed here.

3.1 Condition of Height-ordering

The height of tasks criterion is used for generating the first population. Therefore, the tasks are ordered in ascending order of their heights. Then, according to this order, the tasks are assigned to processors. For example, As shown in Figure 2 that $height(T_1) \leq height(T_5) \leq height(T_4) \leq height(T_7)$.
If there is no relationship between two tasks, then the height-ordering condition will not be used. For example the tasks T_5 and T_6 could be executed in any arbitrary order. The optimal or sub-optimal schedule may not satisfy the height-ordering condition. So, the definition of height could be modified as Equation (2). $height(T_j)$ is a random integer X which could be obtained from Equation 2.

$$\begin{aligned} &\min hight(T_k) - 1 \leq X \leq \text{Max}\ hight(T_i) + 1; \\ &\text{for } \forall\, T_i \in PRED(T_j), T_k \in SUCC(T_j) \end{aligned} \quad (2)$$

3.2 Fitness Value

The GA uses multiple search nodes simultaneously. Each of the search nodes corresponds to one of the current solutions (schedules) and is represented by a chromosome. Each chromosome contains a string, called genes (tasks) and has an associated value called a fitness value, which is evaluated by a fitness function. The fitness function used for the genetic algorithm is based on the total finish time of the schedule which is obtained by Equation 3.

$$\begin{aligned} &Fitness = \max\ \{FT(P_j)\}; \\ &\qquad \text{for } j = 1,2,\ldots,N_P \end{aligned} \quad (3)$$

Where N_p is the number of processors and $FT(P_j)$ is the finish time of the final task in the processor P_j.

3.3 Initial Population Generating

In a GA, only chromosomes with better fitness values are likely to survive in the next generations. By evolving the chromosomes continuously, the solutions corresponding to the search nodes are improved gradually. A set of chromosomes at a given stage of a GA is called a population. The number of chromosomes in a population is called the population size. The following steps randomly create an initial population of a task graph for a multiprocessor with N_p processors.

1- Put the tasks in a list according to their height in ascending order.
2- Repeat step 3 until all the tasks would be finished.
3- Generate an integer number, r, between 1 and N_p, then select the first task in the list and assign it to processor r and then delete it from the list.

4- By repeatedly applying the above steps (for the number of population size), initial population would be generated.

3.4 Crossover operation

The crossover starts with two parent chromosomes to exchange subparts of them to create two new children chromosomes as shown in Figure 3 like following steps.

1- Select the crossover points where the list could be cut into two halves, according to the two next conditions.
2- Exchange the bottom halves of processor P1 in chromosome A and chromosome B.
3- Exchange the bottom halves of processor P_2 in chromosome A and chromosome B.

The crossover is applied with a certain crossover rate (X_r).

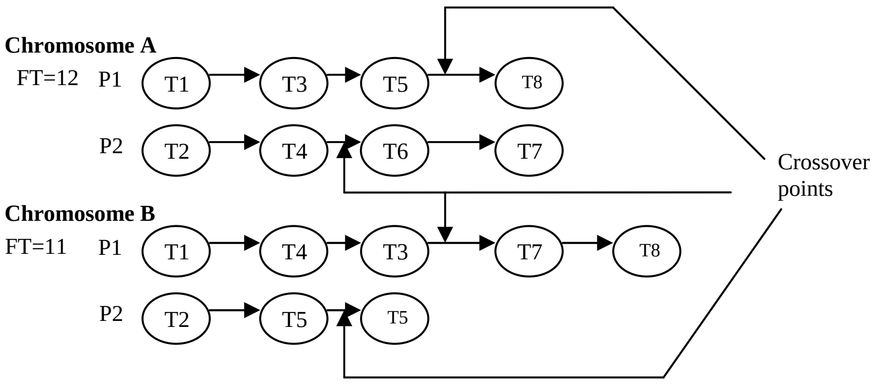

Figure 3- Applying crossover to two chromosomes.

If the crossover points satisfy two following conditions then, the new chromosomes will be legal.

1- The height of the tasks next to the crossover points should be different.
2- The height of all the tasks immediately in front of the crossover points should be equal.

The crossover could be easily extended for N_p processors.

3.5 Mutation Operation

The mutation selects a chromosome and then randomly exchanges the two tasks with the same height. The mutation is applied with a certain mutation rate (M_r) which is used to prevent the search process from converging to the local optima prematurely.

3.6 Selection and Reproduction

The selection is implemented by a biased roulette wheel [7] where each chromosome in the population occupies a slot size in proportion to its fitness value. Each time a generation is required, a simple spin of the biased roulette wheel yields a parent chromosome. Because chromosomes with higher fitness values will have larger slots, they are more likely to be selected and to be prepared for crossover and mutation. Elitism, the property of guaranteeing the best solution passes in the next generation, is used here to improve the selection method.

4 The Proposed Algorithm

We can make several modifications to improve the BGA which is described here.

4.1 The new Initial Population Generating

The bottom-level of a task is the length of the longest path from the task to an exit task (the task has no child). The bottom-level is obtained by two conditions:

- If a task has no child, its bottom-level is equal to its execution time.
- If a task has child, its bottom-level is equal to the maximum bottom-level of its children.

Step 2 and step 4 of the proposed algorithm are new compared to the BGA.

1- Sort the tasks according to their heights in ascending order.
2- Sort the tasks with the same height according to their bottom-level in descending order.
3- Repeat step 4 and step 5 until finish of all the tasks.
4- Generate a permutation of processors.
5- Assign tasks to processors in order.
6- The above steps are repeated for the number of population size.

A decrease in the computation time for the new algorithm is realized for the following reason: Step 2 is not present in the GBA which means there is no priority between tasks with the same height. The bottom-level prioritization orders tasks with equal height in the new algorithm.

For example, Figure 4 shows a DAG and Table 1 represent the priority of tasks' execution based on their height and their bottom-level too.

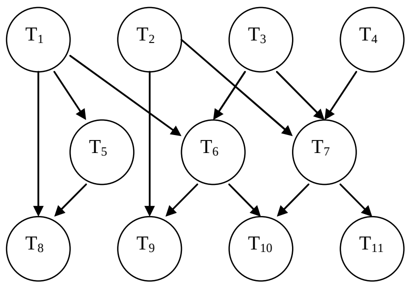

Figure 4- An example of a DAG.

Finally, Figure 5 shows two schedules for three processors based on the different policies.

Table 1- priority of execution of tasks based on their height and their bottom-level.

Task number	Execution time	Height	Bottom-level	Order of execution according to height	Order of execution according to bottom-level
1	50	0	72	1	1
2	1	0	41	2	4
3	10	0	50	3	3
4	20	0	60	4	2
5	20	1	21	5	7
6	2	1	22	6	6
7	20	1	40	7	5
8	1	2	1	8	11
9	20	2	20	9	9
10	19	2	19	10	10
11	20	2	20	11	8

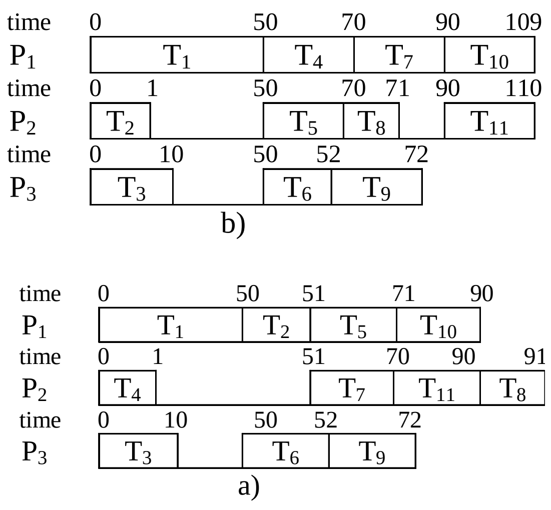

Figure 5- Two schedules for three processors based on the different policies. a) Order of execution according to the bottom-level. b) Order of execution according to the height.

As it seen, by changing the policy (priority) of task assignment, the total finish time based on the priority of the tasks' bottom-level is 91 while that, the total finish time based on the priority of the tasks' height is 110, which is significantly longer. Now, using the tasks' bottom-level is considerably better than using the tasks' height when the number of tasks increases.

In step 4, instead of choosing processors randomly, a permutation of processors is performed. Using of permutation has the following three advantages:

- There are no idle processors after task assignment.
- The tasks are distributed and are balanced across the processors.
- The tasks with the same height are distributed as far as possible.

Considering that, the tasks with the equal height can execute in parallel, so if there are several tasks with the same height for executing in a processor then, the parallel execution will become the sequential execution and the founded schedule will be not optimal or sub-optimal solution.

On the other hand, if the tasks with the same height which is assigned to a processor have some successors then, their successors will start lately and these delays propagate to the task graph and generate a non optimal schedule. This question maybe asked that a load imbalance across processors will occurred only in the initial population and it will disappear in the further generations. But, it can be noticed that it takes some time for the several generations until balancing load among processors to distribute the tasks with the same height to the different processors. This means that, the BGA takes some time to distribute the tasks with equal height to the different processors.

Figure 6 shows a schedule of eleven tasks of Figure 4 on three processors, which the tasks with the same height are not distributed to the different processors. The total finish time of this schedule is 131, which is not good however, the processors are load balanced.

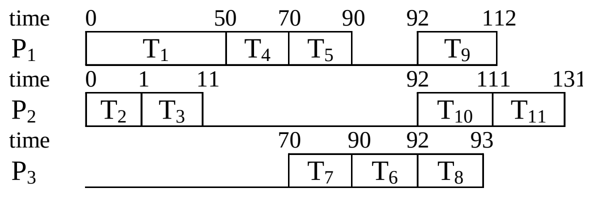

Figure 6- A schedule of tasks of Figure 4 on three processors, which the tasks with the same height are not distributed to the different processors.

4.2 New Selection and Reproduction

1- Sort the chromosomes according to their fitness value in ascending order.
2- Copy the N_{Elite} best chromosomes with the best fitness values to the new generation (the number of N_{Elite} will be discussed later in elitism stepping technique).
3- Use a biased roulette wheel for the current generation.
4- For (the number of population size – N_{Elite})/2 do step 5, 6 and 7.
5- Select two chromosomes by using the biased roulette wheel and prepare them for crossover.
6- With probability X_r apply the crossover to the chromosomes and move the children to the next generation; otherwise, prepare them for mutation.
7- With probability M_r apply the mutation to the two chromosomes and move the generated chromosomes to the next generation; otherwise, they go into the next generation with no change.

Typically, above steps would be terminated after a certain number of generations or if a convergence would be reached.
It can be noticed that the population size of each generation stays fixed by applying the above steps.

4.3 The Elitism Stepping Technique

It is obvious that a fixed elitism with a few elites has a low convergence rate (a high computation time) but it can find a sub-optimal schedule. On the other hand, the fixed elitism with a lot of elites has fewer solutions in the search nodes and can not find a sub-optimal schedule, however it converges fast. Therefore, a new method, named elitism stepping is introduced here which uses the advantages of two mentioned fixed elitism. The aim of this technique is to decrease the computation time of the algorithm to find an acceptable sub-optimal schedule.

In this technique, the number of elites in the first generation is two and after that, by increasing the number of generations, the number of elites increases too, until they reach to the population size. So, the convergence is happened and a sub-optimal schedule is founded.
By performing some simulation of a GA for the different DAGs and by studying diagrams of the average finish time of schedules in different generations, it is seen that the stepping elite has suitable behavior according to the nature of task scheduling problem. Figure 7 shows the average finish time of a GA for a set of 15 random DAGs consist of 20 to 90 tasks for scheduling in 3 to 6 processors where, each simulation has 200 generations.
The average finish time for the first generation is 883.18, for the 100th generation is 747.64 and for 200th generation is 732.28. If the number of generation is divided to three parts then, the first one-third has the most effectiveness to improve the solutions. The second part improves the solution slowly and in the last one-third, the average finish time usually converges to one value.
It is obvious that by using the elitism stepping technique, there are more areas in the search nodes at first because the first solutions has more ability of improvement. Then, as the number of generations increases, the ratio of improvement decreases, so this technique causes fewer areas exist in the search nodes and all solutions converge to a sub-optimal schedule.

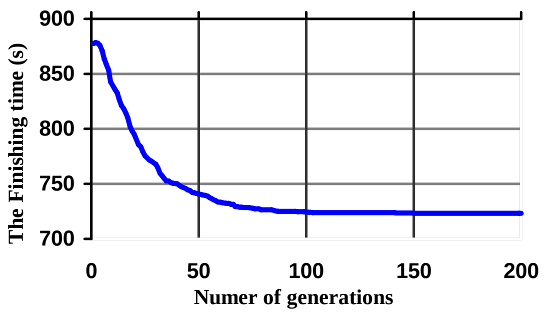

Figure 7- The average finish time for 200 generations.

5 Experimental Results

A set of simulation is performed under one set of common assumptions by MATLAB7 for comparison of the proposed algorithm and the previous BGA. A Pentium IV-2.8MHz with RAM 512M based computer is used to implement both algorithms. A random-graph-generating program is written by C++ language, which is capable of making random graphs that meet initial constraints. For this purpose a set of 15 graphs consists of 20 to 90 tasks with random execution time are generated. These tasks would be scheduled on a multiprocessor system with 3 to 6 processors as shown in Table 2.

Table 2- Specifications of the tasks and the multiprocessors would be scheduled.

Graph number	Number of tasks	Number of processors
1	20	3
2	25	3
3	30	3
4	35	4
5	40	4
6	45	4
7	50	5
8	55	5
9	60	5
10	65	5
11	70	5
12	75	5
13	80	6
14	85	6
15	90	6

After generating the data, the parameters of the both algorithms need to be determined. The number of generation is set to 200, so stop condition is 200 iterations. In order to achieve a proper search nodes and whereas the number of generation is fixed then, the population size is chosen proportionate to the number of tasks and it is set to 1.6 times the number of tasks. The crossover and mutation rates are set to 0.8 and 0.06 respectively.

Each graph in Table 2 is scheduled for both algorithms three times. The average computation time of two algorithms for finding a sub-optimal schedule and their average finish time (fitness values) are calculated for each graph and then, the total average time are obtained for 15 graphs.

The average computation time of two algorithms for 200 generations is shown in Figure 8. The average computation time varies linearly according to the increas in the number of generations in the BGA. Since, the average computation time of generation 200 (a sub-optimal schedule) is 18.45s for BGA and is 10.06s for the proposed algorithm. Therefore, the speed up of the new algorithm is almost 1.85 times of the GBA.

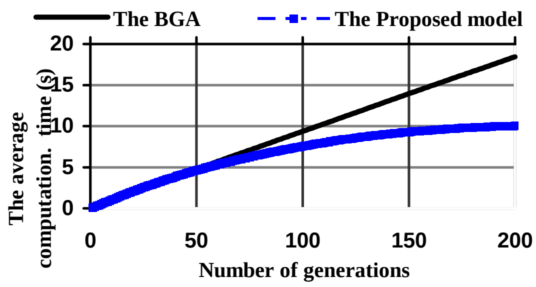

Figure 8- The average computation time of two algorithms for 200 generations.

The average finish time (the average fitness values of schedules) of two algorithms for 200 generations is shown in Figure 9. As it seen, the average finish time of generation 200 in the proposed algorithm is better than GBA.

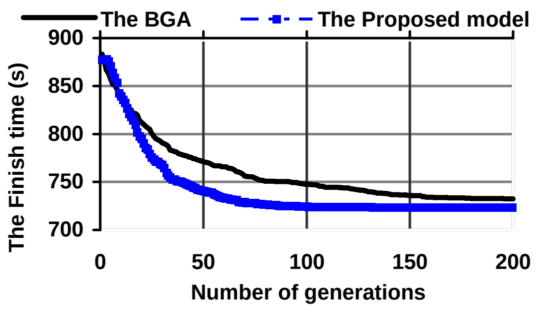

Figure 9- The average finish of two algorithms for 200 generations.

6 Conclusions

In this paper, we presented a genetic algorithm which uses a new method, named elitism stepping technique for the task scheduling problem in multiprocessor systems, with the objective to reduce the schedule length within an acceptable computation time. In order to show the effectiveness of the proposed algorithm, we performed experimental simulations by applying the algorithm to various kinds of task graphs. Additionally, we compared the new algorithm with the previous genetic algorithm, BGA proposed by Hou. As a result, under the same conditions, the proposed algorithm obtains better schedule length or finish time than the BGA. Moreover, the results show that the computation time of the new algorithm is much smaller than the previous one.

References

[1] Andersson B., Baruah S. and Jonsson J., ”Static-Priority Scheduling on Multiprocessors”, In Proc. of the 22nd IEEE Real-Time Systems Symposium, London, England, 2001.

[2] Auyeung, A., Gondra, I. and Dai, H.K. "Multi-heuristic List Scheduling Genetic Algorithm for Task Scheduling", Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, ACM Press, pp. 721-724, 2003.

[3] Bouffard V., Ferland J. A.: Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem. Journal of Scheduling, Vol. 10(4), pp. 375-386, 2007.

[4] Chandra A., Adler M. and Shenoy P., ”Deadline fair scheduling: Bridging the theory and practice of proportionate fair scheduling in multiprocessor systems”, In Proc. of the 7th IEEE Real-Time Technology and Applications Symposium, May 2001.

[5] Dandass, Y. S., “A Genetic Algorithm for Scheduling Acyclic Digraph in the presence of Communication Contention”, In Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications, pp. 223-230, 2003.

[6] Davidovic T. and Crainic T.G. , “Benchmark-Problem Instances for Static Scheduling of Task Graphs with Communication Delays on Homogeneous Multiprocessor Systems”, C.R.T.'s publications, 2004.

[7] Haupt, R.L., Haupt, S.E., Parallel genetic algorithms, John Wiley & Sons, 2004.

[8] Hou E. S. H, Ansari N. and H. Ren, ”A Genetic Algorithm for Multiprocessor Scheduling”, IEEE trans. on parallel and distributed systems. vol. 5, no. 2, pp. 113-120, Feb. 1994.

[9] Lee, Y.H., Chen, C., "A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems", the 9th workshop on compiler techniques for high-performance computing, 2003.

[10] Ramamurthy S. and Moir M., “Static-priority periodic scheduling on multiprocessors”, In Proc. of the IEEE Real-Time Systems Symposium, pp. 69-78, Orlando, November, 2000.

[11] Shenassa, M. H. and Mahmoodi, M., "A Novel Intelligent Method for Task Scheduling in Multiprocessor Systems Using Genetic Algorithm", journal of Franklin Institute, Elsevier, pp. 1-11, 2006.

[12] Srinivasan, A., Anderson, J.H. “Efficient Scheduling of Soft Real-Time Applications on Multiprocessors”, 15th Euromicro Conference on Real-Time Systems (ECRTS'03**)**, Porto, Portugal, 2003.

[13] Yoo M. and Gen M.: Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system. Computers and Operations Research, Vol. 34(10), P. 3084-3098, 2007.

[14] Zafarani Moattar E., Rahmani A.M., Feizi Derakhshi M.R. "Job Scheduling in Multi Processor Architecture Using Genetic Algorithm", 4th IEEE International conference on Innovations in Information Technology, dubai, pp. 248-251, 2007.