An Efficient and Robust Genetic Algorithm for Multiprocessor Task Scheduling

Sachi Gupta, Gaurav Agarwal, and Vikas Kumar

***Abstract*—The general problem of multiprocessor scheduling can be stated as scheduling a task graph onto a multiprocessor system so that schedule length can be optimized. Task scheduling in multiprocessor system is a NP-complete problem. In literature, several heuristic methods have been developed that obtain suboptimal solutions in less than the polynomial time. Recently, Genetic algorithms have received much awareness as they are robust and guarantee for a good solution. In this paper, we have developed a genetic algorithm based on the principles of evolution found in nature for finding an optimal solution. Genetic algorithm is based on three operators: Natural Selection, Crossover and Mutation. To compare the performance of our algorithm, we have also implemented another scheduling algorithm HEFT which is a heuristic algorithm. Simulation results comprises of three parts: Quality of solutions, robustness of genetic algorithm, and effect of mutation probability on performance of genetic algorithm.**

***Index Terms*—Genetic algorithm, fitness function, multi-processor system, NP-complete etc.**

I. Introduction

Task scheduling in multiprocessor systems also known as multiprocessor scheduling has been a source of challenging problems for researchers in the area of computer engineering. The general problem of multiprocessor scheduling can be stated as scheduling a set of partially ordered computational tasks onto a multiprocessor system so that a set of performance criteria will be optimized.

Unfortunately, Task scheduling in multi processor system is NP-complete [1]. There are various methods by which we can find the solution of scheduling problems in less than the polynomial time. Heuristics based scheduling methods are on of them but it has seen that they give good results for some inputs while bad for others. One another recent solution of scheduling problem is Genetic algorithms (GA), which are based on the mechanics of natural selection and natural genetics. The main goal behind research on genetic algorithms is robustness. Genetic algorithms [2]-[5] are the most popular random search techniques for different kind of task scheduling problems.

A genetic algorithm continuously tries to improve the average fitness of a population by construction of new populations. Quality of solution depends heavily on the selection of some key parameters like fitness function, population size, crossover probability and mutation probability.

In this paper, we first introduce task scheduling problem having some specified characteristics, after that genetic approach is discussed in detail and the last section presents experiments and results.

II. Multiprocessor Task Scheduling

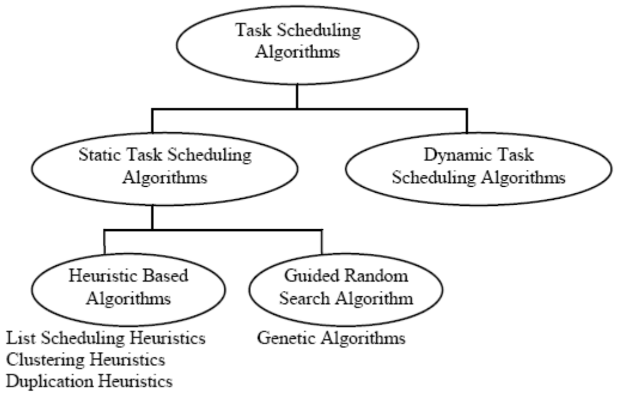
Many parallel applications consist of multiple functional units. While the execution of some of the tasks depends on the output of the other tasks, others can be executed independently at the same time, which increases parallelism of the problem. The task scheduling problem is the problem of assigning the tasks in the multiprocessor system in a manner that will optimize the overall performance of the application, while guarantee the correctness of the result. Multiprocessor scheduling problems can be classified into many different categories based on characteristics of the program and tasks to be scheduled, the multiprocessor system, and the availability of information (See Fig. 1). The two main categories of Multiprocessors Task scheduling are: Static and dynamic task scheduling. A static or deterministic task scheduling is one in which precedence constraints and the relationships among the task are known well in advance while non-deterministic or dynamic scheduling is one in which these information is not known in advance or not known till run time. Here we only consider static task scheduling problems. Static task scheduling algorithms [6], [7] can be classified into two parts: Heuristic Based and Guided random Search Based Algorithms. Heuristic based algorithm searches a path in the solution space based on the heuristic used while ignoring other possible paths. List scheduling algorithms [8], clustering [8] and duplication based algorithms [9], [10] fall under this category.

In List Scheduling Heuristic, each task is allotted a priority then added to a queue of waiting tasks in order of decreasing priority. As processors become available, the task with the highest priority is deleted from the queue and assigned to the most suited processor. The major difference between algorithms in this category is the way by which priorities are assigned and the most suited processor is allocated. In Clustering Heuristic, tasks of a given task graph are mapped into an unlimited number of clusters. In this heuristic, each iteration refines the previous clustering by merging some clusters. If two tasks are assigned to the same cluster, they will be executed on the same processor. In duplication based heuristic, scheduling of a task graph is done by mapping some of its task redundantly, which reduces the inter process communication over head.

Manuscript received September 15, 2012; revised November 30, 2012.

S. Gupta and G. Agarwal are with the Computer Science and Information Technology Deptt. Krishna Institute of Management and Technology, Moradabad, India (e-mail: sachiagarwal@rediffmail.com, meetgaurav1@rediffmail.com)

V. Kumar is with the Computer Science Deptt, Moradabad Institute of Technology Moradabad, India (e-mail: Vikas_in_mittal@rediffmail.com).



Task Scheduling Algorithms
Static Task Scheduling Algorithms
Dynamic Task Scheduling Algorithms
Heuristic Based Algorithms
Guided Random Search Algorithm
List Scheduling Heuristics
Clustering Heuristics
Duplication Heuristics
Genetic Algorithms

Fig. 1. Classification of task scheduling algorithm

Guided random search techniques use random choices to guide them selves through the problem space. Genetic Algorithm [11], [12], [13] Belong to the Guided random Search based category.

III. Problem Description

In our work, we considered Static scheduling problems with the following characteristics:

- Tasks are non preemptive in nature. Precedence relations among the tasks exist.
- Communication costs do not exist.
- In a Multiprocessor System, all the processors are heterogeneous meaning thereby a task may take different execution time on each processor.

A static scheduling problem consists of three main components: A multiprocessor system, an application and an objective for scheduling.

The multiprocessor system consists of a limited number of fully connected heterogeneous processors (P_1, P_2... P_m). An application comprises tasks and their dependencies on each other. It can be represented as a directed acyclic graph (DAG) [14], [15], (see Fig.2) $G = (V, E, W)$, where the vertices set V consists of v non preemptive tasks, and v_i denotes the i^{th} task. The edge set E represents the precedence relationships among tasks. A directed edge e_{ij} in E indicated that v_j can not begin its execution before receiving data from v_i. W is a matrix of vxm, and w_{ij} in W represents the estimated execution time of v_i on j^{th} processor.

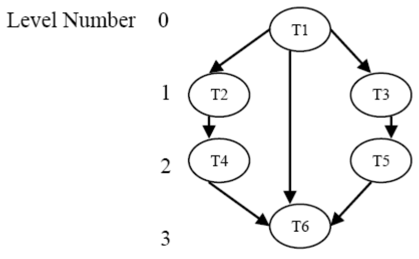


Fig. 2. A sample directet acyclic graph

The main objective of the task scheduling is to determine the assignment of tasks of a given application to a given parallel system such that the execution time (or schedule length) of this application is minimized satisfying all precedence constraints.

IV. Genetic Approach

Genetic Algorithms or evolutionary algorithms are developed by John Holland in 1960s. They are random search based algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GA is designed to simulate processes in natural system necessary for evolution. Some key terms used in genetics are as follows:

- Gene: A single encoding of a part of the solution space.
- Chromosome: A string of “Genes” that represents a solution.
- Population: The number of “Chromosomes” available to test.
- Locus: A unique position a gene can occupy on the chromosome.

Genetic Algorithm uses three operators known as natural selection, crossover and mutation. Genetic algorithm differs from the traditional optimization methods in the following ways:

- Genetic Algorithms are both effective and robust.
- They can produce high quality solutions.
- Able to exploit favorable characteristics of previous solution attempts to construct better solutions.
- Computationally simple and easy to implement.
- Genetic Algorithms use probabilistic transition rules, not deterministic rules.

A. *Genetic Algorthtm Structrure*

The structure of a genetic algorithm for any problem depends on five things which are as follows:

- The choice of representation of chromosomes.
- Construction of genetic operators.
- The choice of Fitness Function.
- Probabilities that can control genetic operators.
- No. of Generations.

Each of the above five parts greatly affects the solution obtained as well as the performance of the genetic algorithm.

Typically, a Genetic Algorithm Structure consists of the following steps:

- GA1: Initialization – initialize the population.
- GA2: Evaluation – evaluate each chromosome using fitness function.
- GA3: Genetic operations –Select the parent and apply genetic operators on them to produce new chromosomes (offspring).
- GA4: Repeat steps GA2 and GA3 until termination condition reached.

From the above steps, we can see that genetic algorithms utilize the concept of survival of the fittest; passing “good” chromosomes to the next generation, and combining different strings to explore new search points.

B. *Initial Population (Structure of the Chromosome)*

Designing of chromosome structure is crucial for devising Genetic Algorithm.

We define our chromosome structure as a combination of two strings SQ and SP, whose length equal to the number of tasks. SQ (scheduling queue) maintains precedence constraints between tasks, and an entry in SQ represents a task to be scheduled. An entry in SP (scheduling processor) represents the processor the corresponding task is scheduled onto.

The details to generate a chromosome can be seen in following steps:

- IP1: Select randomly a task from the entire entry tasks. Set this task as the first task in SQ.
- IP2: Repeat step IP3 for (v-1) times.
- IP3: Randomly select a task who is not in SQ and whose predecessors all have been in SQ, and add this task to SQ.
- IP4: For SP part, randomly generate an integer number between 1 and m for each task in SQ and add it to SP.

C. Evaluation and Selection: Roulette Wheel Mechanism

The fitness function in genetic algorithms is typically the objective function that we want to optimize in the problem. It is used to evaluate the chromosomes. For calculating the fitness values and in order to select good chromosomes, we define the fitness function as:

$$F\ (i) = (\max FT\text{-}FT\ (i)\ +1/\ (\max FT\text{-}\min FT\text{+}1) \qquad (1)$$

where: maxFT and minFT is the maximum and minimum finishing time of chromosomes in current generation, respectively. $FT(i)$ is the finishing time of the i^{th} chromosome.

Once the fitness values of all the chromosomes have been evaluated, we can select the higher fitness value chromosomes using the roulette wheel mechanism. We have implemented a roulette wheel mechanism where each chromosome in the population occupies a slot size proportional to its fitness value. Random numbers are generated and used as an index into the roulette wheel to determine which chromosome will be passed to the next generation. Because chromosomes with higher fitness value will have larger slots, they are more likely to be selected and passed to the next generation.

D. Reproduction: Crossover and Mutation

Crossover

Crossover is a mechanism that produces new offspring that have some parts of both parent's genetic material. The simplest type of crossover is single-point crossover. Multipoint crossover uses m randomly chosen crossover positions. Bits between successive crossover points are exchanged producing two new offspring.

As our chromosome comprises two separate parts SP and SQ having different characteristics, for each part we employ different crossover policies. We randomly select one or the second part and apply two different crossover operators for these two parts.

Details about crossover are given in following steps:

- CR1: Input the Crossover probability P_c.
- CR2: Randomly select pairs of chromosomes and generate a float number (FLC) between 0 and 1 for each pair.
- CR3: If $FLC <= P_c$, then repeat step CR4 to step CR5 Else directly reproduce those two chromosomes to the next generation.
- CR4: Randomly generate two crossover points, p and q, between 1 and v and crossover flag CF between 0 and 1.
- CR5: If CF=0 then rearrange the order of tasks in SQ between p and q of one chromosome according to the order of tasks of another chromosome, the rest of the two chromosomes are remained. Else exchange the part in SP between p and q of two chromosomes and the rest of the two chromosomes are remained.

Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new chromosomes being added to the population. With these new chromosomes, the genetic algorithm may be able to achieve a better solution than was previously possible. Mutation can be considered as a random alternation of the individual.

We employ two policies to mute the chromosome as given in following steps:

- MT1: Input the Mutation probability P_m.
- MT2: For each chromosome, generate a float number (FLM) between 0 and 1.
- MT3: If $FLM <= P_m$, then repeat step MT4 to step MT5 Else directly reproduce this chromosome to the next generation.
- MT4: Randomly generate a mutation point p between 1 and v and mutation flag MF between 0 and 1.
- MT5: If MF=0 then select randomly a location between location of the nearest immediate predecessor and that of successor of sq_p. Then move sq_p to this location. Else change randomly the processor of sq_p between 1 and m as sp_p.

V. Experiments and Results

In our work, we implemented two algorithms for solution of multiprocessor task scheduling problem. One is based on list scheduling heuristic HEFT and other is our proposed Genetic Algorithm. For performance evaluation of our algorithm we generated some problems of varying sizes and solved them by both the algorithms. Details of our experimental setup and results obtained by HEFT and proposed GA are as given below.

A. Experimental Setup

We have implemented a system to automatically generate the scheduling problems of required sizes. This we have done to avoid biasing in giving values of different parameters required for the problems. Our system fits random values to these parameters in appropriate ranges. We have generated problems for our experiments with the following characteristics:

- Size of problem ranges from 25 to 65 with an interval of 5.
- There is no limit on the number of successors of each task except the exit task which does not have any successor.
- The execution time for each task is a random number between 5 and 25.
- Number of processors varies from 4 to 8 according to the size of problems.

As we did not put any restriction over the number of successor a task may have, task graph may be much complicated. So, the problems we have chosen may be considered difficult in comparison to the kind of problems we normally see in literature, where a restriction on maximum number of successor tasks has been put.

B. Results of HEFT

The Heterogeneous-Earliest-Finish-Time (HEFT) Algorithm is a heuristic scheduling algorithm for a bounded number of heterogeneous processors, which has two major phases: a task prioritizing phase for computing the priorities of all tasks and a processor selection phase for selecting the tasks in the order of their priorities and scheduling each selected task on its "best" processor, which minimizes the task's finish time.

We run HEFT procedure on ten different problems with Problem Identification Numbers (PIN) 0 to 9 for each problem size to note the length of the schedules obtained (see Table I). We then computed average schedule length for each problem size for comparison with corresponding results obtained from GA.

TABLE I: RESUTS OF HEFT

Size of Problem	**PIN (0-9)**										**Average Schedule Length**
	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	
25	116	108	145	105	107	144	141	154	148	170	133.8
30	158	165	170	140	170	167	176	147	161	173	162.7
35	161	163	162	193	139	176	140	177	192	197	170
40	203	183	154	180	202	208	174	184	162	211	186.1
45	217	184	193	186	229	179	171	216	256	217	204.8
50	192	220	186	241	236	201	203	249	218	216	216.2
55	233	220	216	240	229	243	240	235	219	237	231.2
60	252	260	259	266	244	219	264	260	247	252	252.3
65	277	294	250	255	275	260	282	284	272	245	269.4

TABLE II: RESULTS OF GA

Size of Problem	**PIN (0-9)**										**Average Schedule Length**
	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	
25	116	108	145	104	107	143	141	154	148	170	133.6
30	158	165	170	140	170	164	176	147	159	173	162.2
35	160	163	162	193	139	176	140	177	191	197	169.8
40	202	180	154	177	202	208	174	183	159	210	184.9
45	217	180	191	186	229	179	171	216	256	216	204.1
50	192	220	186	241	236	201	203	249	217	216	216.1
55	233	220	215	239	230	243	240	235	219	237	231.1
60	252	259	256	266	246	220	264	260	247	252	252.2
65	277	294	249	253	273	261	281	284	271	244	268.7

TABLE III: COMPARISON OF HEFT AND GA

No. of tasks	**No. of processors**	**Avg. schedule length (HEFT)**	**Avg. schedule length (GA)**
25	4	133.8	133.6
30	4	162.7	162.2
35	5	170	169.8
40	5	186.1	184.9
45	6	204.8	204.1
50	6	216.2	216.1
55	7	231.2	231.1
60	7	252.3	252.2
65	8	269.4	268.7

A. *Results of GA*

The proposed genetic algorithm was implemented and evaluated on the same set of problems we used to evaluate HEFT. Results obtained are shown in Table II. We set parameters for our Genetic Algorithm as:

- Population Size=25,
- Maximum Generations= 5000,
- Crossover Probability= .6
- Mutation Probability =.2

B. *Comparison of HEFT and GA*

Results obtained from our experiments are analyzed for following factors:

Quality of solution: Comparison of average schedule length of the GA and HEFT is given in Table III and in Fig. 3. Results demonstrate that our proposed Genetic Algorithm is able to compete with heuristic based algorithms as far as quality of solution is concerned. As heuristics are biased towards certain characteristics of solution so they tend to search solution only in a small part of whole search space. It is also possible that they never explore a particular region of search space. Thus for some problems heuristics may give bad results also if they are not chosen carefully.

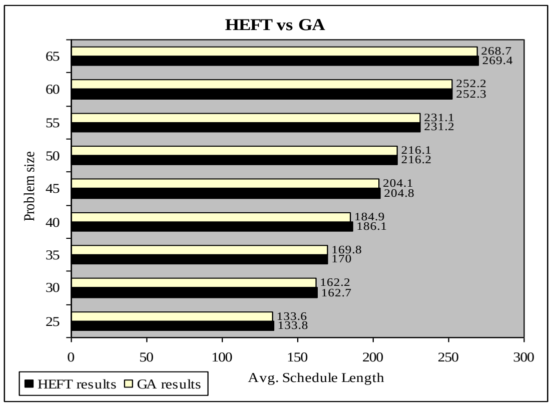


Fig. 3. HEFT Vs GA

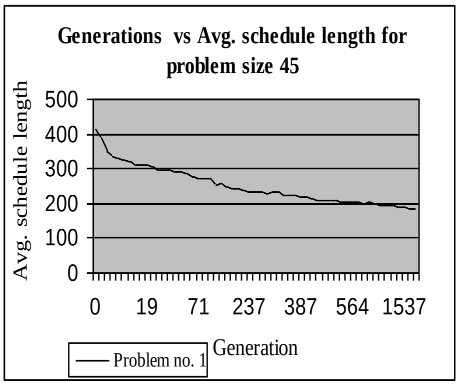


Fig. 4. Problem Size vs avg. schedule length for problem size 45

On the other hand GA is a more powerful method as it searches simultaneously in many parts of search space. Because of mutation operator, change in region being searched, gives potential to GA to search in any part of the search space. Thus it is more likely to find a better or best solution.

Robustness and gurantee for good solution: During our experiments on GA we noted Average schedule lengths of populations emerging generations after generation (see Fig. 4).Though we have shown results only for problem size 45 in Fig. 4, for each problem irrespective of its size we observe that average schedule length is continuously decreasing as more and more generations are evolving. This shows that Genetic Algorithm is robust and ultimately it will give us a good quality solution as quality of solution set is continuously improved. It also reveals that more generations we evolve; it is likely to have better quality in solution.

Effect of mutation probability on the performance of GA: As mutation is the key to change the region of search space, mutation probability may have dominating role in finding solutions of good quality. Thus, we repeated our experiments by fixing crossover probability and changing mutation probabilities from 0.05 to .40 and noted average schedule lengths. We done our experiments on the problem having size 65. we can observe the similar trend in the problems of all sizes. Fig. 5 shows the further average of results, mixing the effect of all crossover probabilities which clearly shows that up till mutation probability is .20, increase in mutation probability leading to better results. After .20 results are fluctuating in a small range but normally are not better than that we obtained for .20. So, we have found best mutation probability for our set of problems as .20.

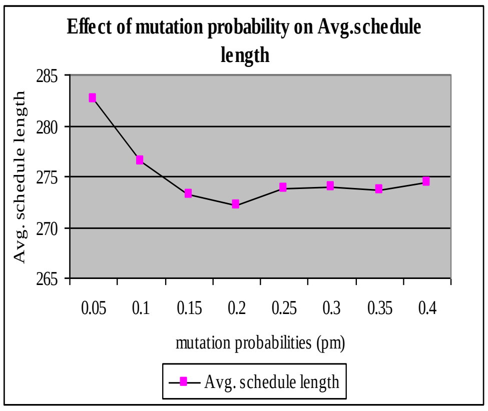


Fig. 5. Effect of mutation probability on avg. schedule length

VI. Conclusion

We have constructed a system which automatically generates the problems of required sizes and also fits values to the parameters. During the experiments with these problems we have seen that for 28.8% problems GA gives lower schedule length, for 4.44% problems GA gives slightly higher schedule length while for 66.67% problems GA gives equal schedule length in comparison with HEFT.

We analyzed performance of our algorithm for robustness. We have seen that in GA, Average Schedule Length continuously decreases as the number of generation increases. This shows that genetic algorithm is robust and gives guarantee for good results.

Lastly, we analyzed effect of mutation probability on the performance of GA. We found that mutation is an escape mechanism to avoid premature convergence. Mutation can be considered as an occasional (with small probability) random alternation of the value of a string. For our problem set, we found best mutation probability as .20.

ACKNOWLEDGMENT

We would like to thank the anonymous referees for their helpful comments and suggestions that have improved the quality of this manuscript. I would also like to express sincere gratitude to Krishna Group for the sponsorship.

REFERENCES

[1] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP completeness," *San Francisco, CA, W. H. Freeman*, 1979.

[2] A. Chipperfield and P. Flemming, "Parallel genetic algorithms, parallel and distributed computing handbook," *First ed., New York*, McGraw-Hill, vol. 143, pp. 111-118, 1996.

[3] D. C. Goldberg, "Genetic algorithms in search, optimization and machine learning," *Add. Wesley Publishing*, 1989.

[4] J. H. Holland, "Adaptation in natural and artificial systems," *MIT Press*, 1975.

[5] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey computer," vol. 27, pp. 17-26, 1994.

[6] H. E. Rewini, T. G. Lewis, and H. H. Ali, "Task scheduling in parallel and distributed systems," *Prentice Hall*, 1994.

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

[8] H. Topcuoglu and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed System*, vol. 13, pp. 260-274, 2002.

[9] B. Kruatrachue and T. G. Lewis, "Duplication scheduling heuristic, a new precedence task scheduler for parallel systems," *Technical Report*, Oregon State Univ., 1987.

[10] T. Tsuchiya, T. Osada, and T. Kikuno, "Genetic-based multiprocessor scheduling using task duplication," *Microprocessors and Microsystems*, vol. 22, pp. 197-207, 1998.

[11] P. C. Wang and W. Korfhage, "Process scheduling using genetic algorithms," in *IEEE Symp. on Parallel and Dist. Proc.*, Texas, USA, Oct. 1995, pp. 638-641.

[12] R. C. Correa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Transactions Parallel and Distributed Systems*, vol. 10, pp. 825, 1999.

[13] Y. W. Zhongiz and J. G. Yang, "A genetic algorithm for tasks scheduling in parallel multiprocessor systems," *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, Xi'an, pp. 2-5 November 2003.

[14] J. L. Gross and J. Yellen, *Handbook of Graph Theory*, CRC Press.

[15] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Second Edition, Elsevier, 2004.