International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 2, Issue 11 (November 2013), PP. 10-18

Task Scheduling in Multiprocessor System using Genetic Algorithm

Atul Kumar Rai[1], Ravindra Gupta[2], Gajendra Singh Chandel[3]
[1]*M. Tech. (SE), SSSIST,SEHOR, MP India.*
[2]*Dept. of Computer Science, SSSIST, SEHOR, MP India.*
[3]*Dept. of Computer Science, SSSIST, SEHOR, MP India.*

Abstract:- 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. In our work we considered only Static scheduling problems with the some characteristics like tasks are non preemptive in nature, precedence relations among the tasks exist, Communication costs do not exist, and all the processors are heterogeneous. Task scheduling in multiprocessor system is NP-complete problem. Various heuristic methods have been proposed that obtain suboptimal solutions in polynomial time. However, a heuristic algorithm may work very well for some inputs, but very poorly for others. We would like a scheduling algorithm to be robust, giving good results regardless of the structure of the task it is to schedule. To develop a scheduling algorithm which tries to find an optimal solution and is robust, we use genetic algorithms. A genetic algorithm applies the principles of evolution found in nature to the problem 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 (Heterogeneous Earliest Finish Time) which is a heuristic algorithm. Our results are divided into three parts: in first part comparison of HEFT and GA demonstrate that our proposed Genetic Algorithm is able to compete with heuristic based algorithms as far as quality of solution is concerned. In second part we observe that irrespective of problem size Average Schedule Length is continuously decreasing as the number of generations increases which guarantee for a good solution. In third part, we observe the effect of mutation probability on quality of solution and found best quality of solution for our set of problems at mutation probability 0.20.

Keywords:- HEFT, Genetic Algorithm, Genetic Operator, Mutation.

I. INTRODUCTION

Task Scheduling in multiprocessor system 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.

We take a deterministic scheduling problem [26] with the following characteristics:

2. Tasks are non preemptive in nature. Precedence relations among the tasks exist.
3. In our work we assume that Communication costs do not exist.
4. The multiprocessor system consists of a limited number of fully connected processors.
5. All the processors are heterogeneous processor.

1.1 Problem Statement Its Possible Solutions

A scheduling problem consists of a multiprocessor computing system, a parallel application and an objective function for scheduling. The multiprocessor computing system consists of a set of m homogeneous processors P= {P_1, P_2... P_m} which are fully connected with each other via identical links. The parallel application can be represented by a directed acyclic graph (DAG), 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. In this case, v_i is called an immediate predecessor of v_j, while v_j is called an immediate successor of v_i. W is a matrix of $v x m$, and w_i, in W represents the estimated execution time of v_i on j^{th} processor.

In a given DAG, a task without any predecessor is called an entry task and a task without any successor is called an exit task. The main objective of the task scheduling [1] is to determine the assignment of tasks of a given application to a given parallel system such that the execution time of this application is minimized satisfying all precedence constraints.

Since this scheduling problem is known to be NP Complete [20], various heuristic approaches have

been developed to solve the problem each with varying degrees of success [4, 8, 14 and 18]. Such methods include heuristic algorithms [8], critical path techniques [18] and acyclic and cyclic graph methods.
An alternative, more recent approach has involved the application of Genetic Algorithms (GA) to multiprocessor task scheduling problems [1, 7].

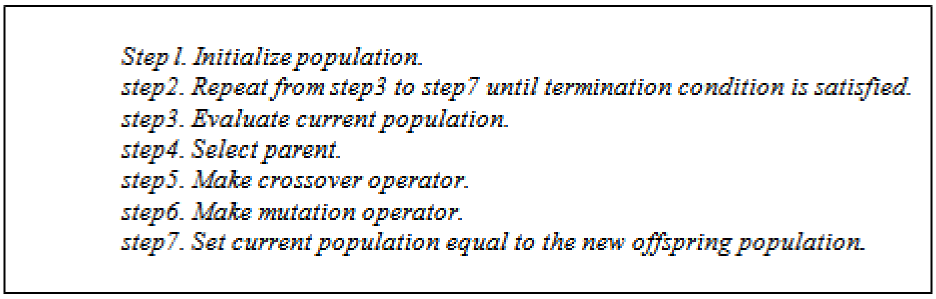
1.3 Aim and Overview of This Thesis

We are trying to develop a genetic algorithm (GA) approach to the problem of task scheduling for multiprocessor systems. Our approach requires minimal problem specific information. Key features of our system include a flexible, adaptive problem representation and an effective fitness function.

This Thesis is organized as follows. Chapter 2 presents the brief introduction of genetic algorithms and strength of genetic algorithm. The system architecture and a method for generating initial population, discussion on the fitness function, and construction of three genetic operators: Natural selection, crossover and mutation are presented in chapter 3. Chapter 4 presents the experiments performed, results obtained and their analysis. Chapter 5 gives the conclusion we can draw from this work and some ideas to extend this work in future.

II. GENETIC ALGORITHM

A genetic or evolutionary algorithm [23] applies the principles of evolution found in nature to the problem for finding an optimal solution. In a "genetic algorithm", the problem is encoded in a series of bit strings that are manipulated by the algorithm. In an "evolutionary algorithm", the decision variables and problem functions are used directly. GAs are based on the adaptive processes of natural systems which are essential for evolution, using direct analogies of natural behavior such as 'populations' of 'chromosomes', 'reproduction', 'cross-breeding' and 'mutation'. They have been shown to be robust stochastic searching algorithms for a wide range of problems. The outline of a GA is described in Fig. 2.1



Step l. Initialize population.
step2. Repeat from step3 to step7 until termination condition is satisfied.
step3. Evaluate current population.
step4. Select parent.
step5. Make crossover operator.
step6. Make mutation operator.
step7. Set current population equal to the new offspring population.

Fig. 2.1: Structure of Genetic Algorithm

2.1 Why GA?

GAs are computationally simple and easy to implement.

- Their power lies in the fact that as members of the population mate and produce offspring, offspring have a significant chance of inheriting the best characteristics of both parents.
- They are able to exploit favorable characteristics of previous solution attempts to construct better solutions.

III. SYSTEM OVERVIEW

We can define our algorithm in following phase:

- Problem Description
- DAG Representation
- Initial Population (Structure of the Chromosome)
- Evaluation and Selection: Roulette Wheel Mechanism
- Reproduction: Crossover and Mutation
- Output of the algorithm (Simulation Results)

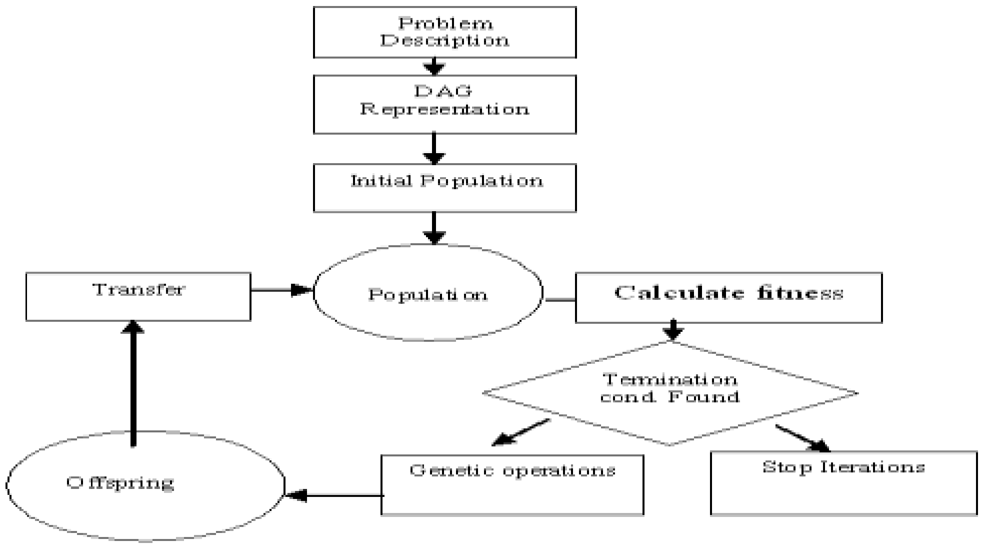


Fig 3.1: System Architecture

3.1 Problem Description

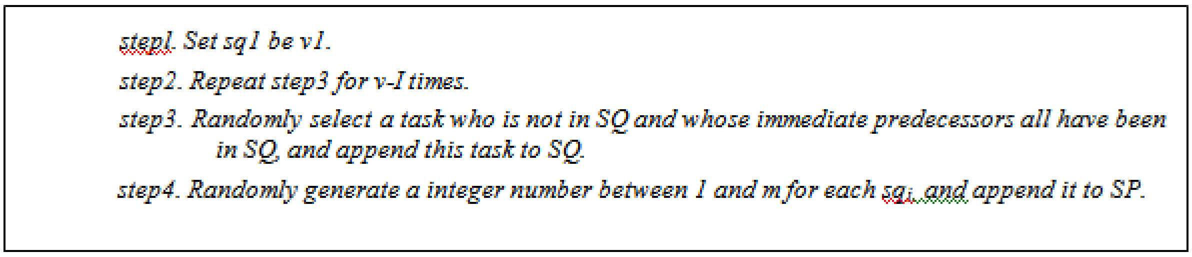
In this Phase, We have implemented a system to automatically generate the scheduling problems of varying sizes.

3.2 DAG Representation

Directed acyclic graphs [6, 17] are directed graphs with no cycles. They are an important class of graphs, being part tree, part graph, and having many applications, such as those involving precedence among "events”. A graph can be represented in computer memory by the adjacency matrix.

3.3 Initial Population

Design of chromosome is crucial for devising GA as they code the solution. A good coding scheme will benefit operators and make the object function easy to calculate. For this purpose, we define a chromosome as two strings {SQ, SP}, who’s length equal to the number of tasks. SQ (scheduling queue) is used to ensure the precedence constraints between tasks, and sq_i in SQ represents the i^{th} task to be scheduled. Each element sp_i in SP (scheduling processor) represents the processor the corresponding task is scheduled onto.
To solve the problem of task scheduling, we generate the initial population just according the precedence constraints between tasks. As a result, any feasible solution may be generated and contained in the initial population.



```
step1. Set sq1 be v1.
step2. Repeat step3 for v-1 times.
step3. Randomly select a task who is not in SQ and whose immediate predecessors all have been
        in SQ, and append this task to SQ.
step4. Randomly generate a integer number between 1 and m for each sqi, and append it to SP.
```

Fig 3.2: The Pseudo Code to Generate a Chromosome

3.4 Evaluation and Selection

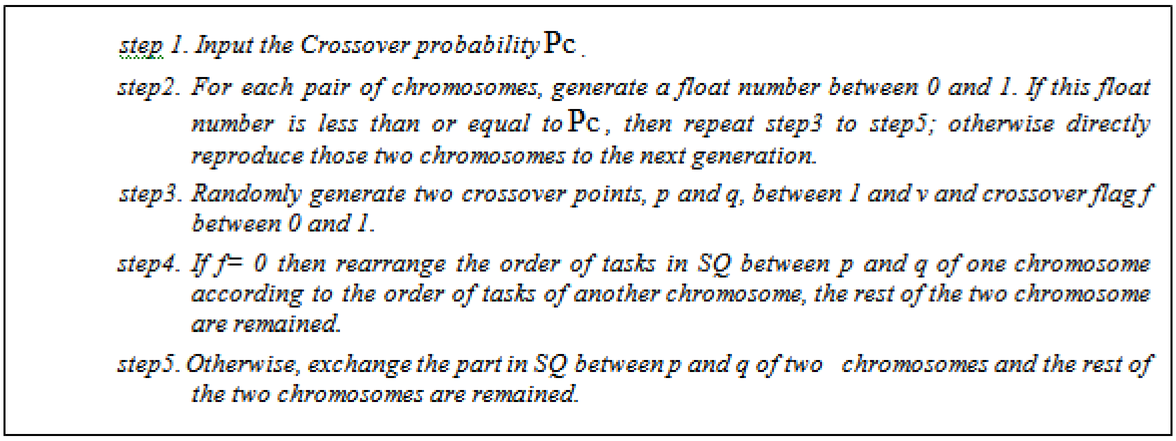
In order to select good chromosomes, we define the fitness function as
$F(i) = (maxSL - SL(i) + 1) / (maxSL - minSL + 1)$ Where: maxSL and minSL is the maximum and minimum finishing time of chromosomes in current generation, respectively. SL (i) is the finishing time of the i^{th} chromosome.

Once fitness values have been evaluated for all chromosomes, we can select good chromosomes through rotating roulette wheel. The chromosomes with higher fitness values have more chance to be selected. And we always save the chromosome with the best fitness so far in current generation to the next generation.

3.5 Crossover

We employ two-cut-point crossover. Because each chromosome consists of two parts with different characteristics, we exploit two crossover operators for these two parts of chromosomes and randomly crossover the first part or the second part.

For the first part SQ, we rearrange the order of tasks in crossover part of one chromosome according to another chromosome. And for the second part SP, we just exchange the crossover subpart of two chromosomes. Details about crossover can be seen in Fig. 3.3.

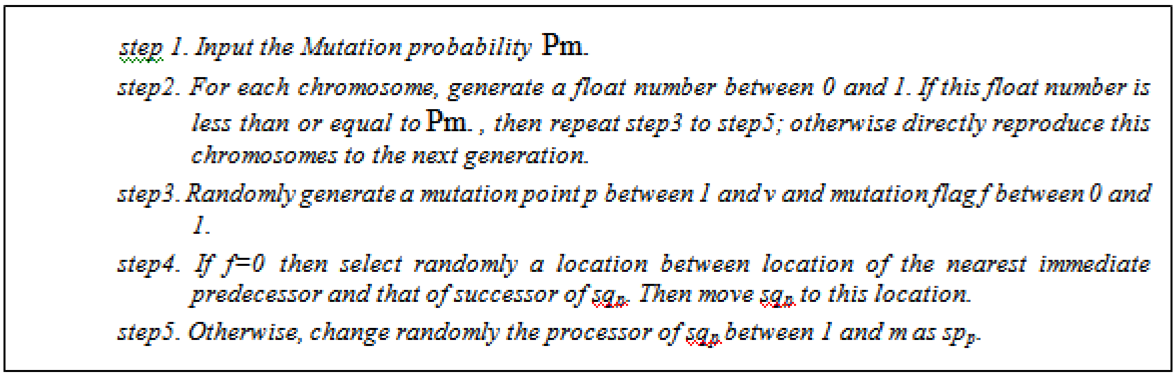


```
step 1. Input the Crossover probability Pc .
step2. For each pair of chromosomes, generate a float number between 0 and 1. If this float
       number is less than or equal to Pc , then repeat step3 to step5; otherwise directly
       reproduce those two chromosomes to the next generation.
step3. Randomly generate two crossover points, p and q, between 1 and v and crossover flag f
       between 0 and 1.
step4. If f= 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 chromosome
       are remained.
step5. Otherwise, exchange the part in SQ between p and q of two chromosomes and the rest of
       the two chromosomes are remained.
```

Fig 3.3: The pseudo code of crossover operator

3.6 Mutation

Mutation can be considered as a random alternation of the individual. We employ two policies to mute the chromosome as shown in Fig. 3.4.



```
step 1. Input the Mutation probability Pm.
step2. For each chromosome, generate a float number between 0 and 1. If this float number is
       less than or equal to Pm. , then repeat step3 to step5; otherwise directly reproduce this
       chromosomes to the next generation.
step3. Randomly generate a mutation point p between 1 and v and mutation flag f between 0 and
       1.
step4. If f=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.
step5. Otherwise, change randomly the processor of $sq_p$ between 1 and m as $sp_p$.
```

Fig 3.4: The pseudo code of mutation operator

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.

4.1 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.

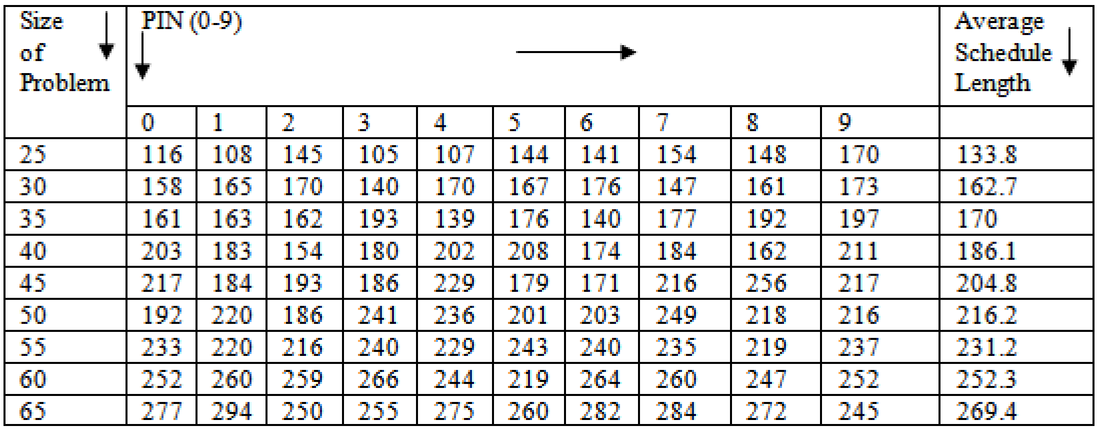
- The execution time for each task is a random number between 5 and 25.
- Number of processors varies from 4 to 8 according to the size of problems.

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.

4.2 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 4.1). We then computed average schedule length for each problem size for comparison with corresponding results obtained from GA.



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

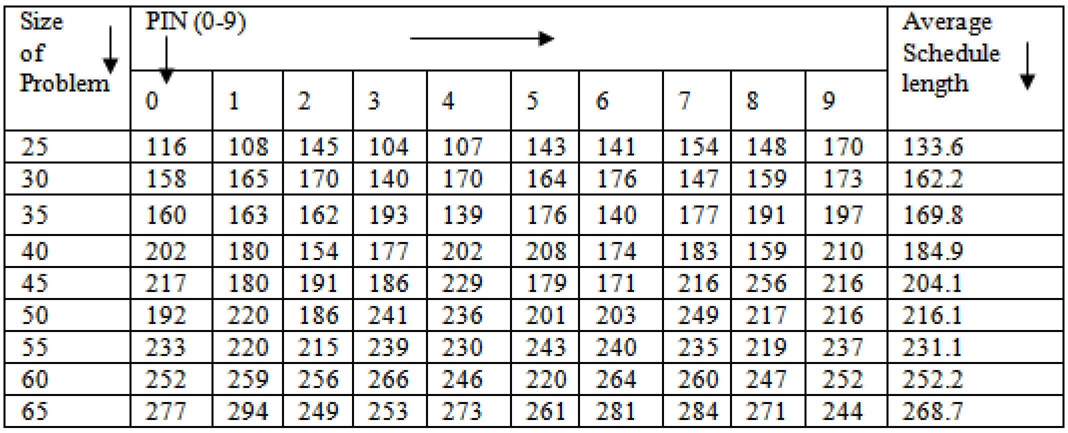
Table 4.1: Results of HEFT

4.3 Results of GA

The proposed genetic algorithm discussed in previous chapter was implemented and evaluated on the same set of problems we used to evaluate HEFT. We set following parameters for our Genetic Algorithm:

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

Results obtained are shown in Table4.2.



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 4.2: Results of GA

4.2 Comparison of HEFT and GA

Results obtained from our experiments are analyzed for following factors:

4.5.1 Quality of solution

Comparison of average schedule length of the GA and HEFT by using different number of processors is given in Table 4.3 and in Fig. 4.4.

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

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.

No. of tasks	No. of processors	Average schedule length (HEFT)	Average 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

Table 4.3: Comparison of HEFT and GA

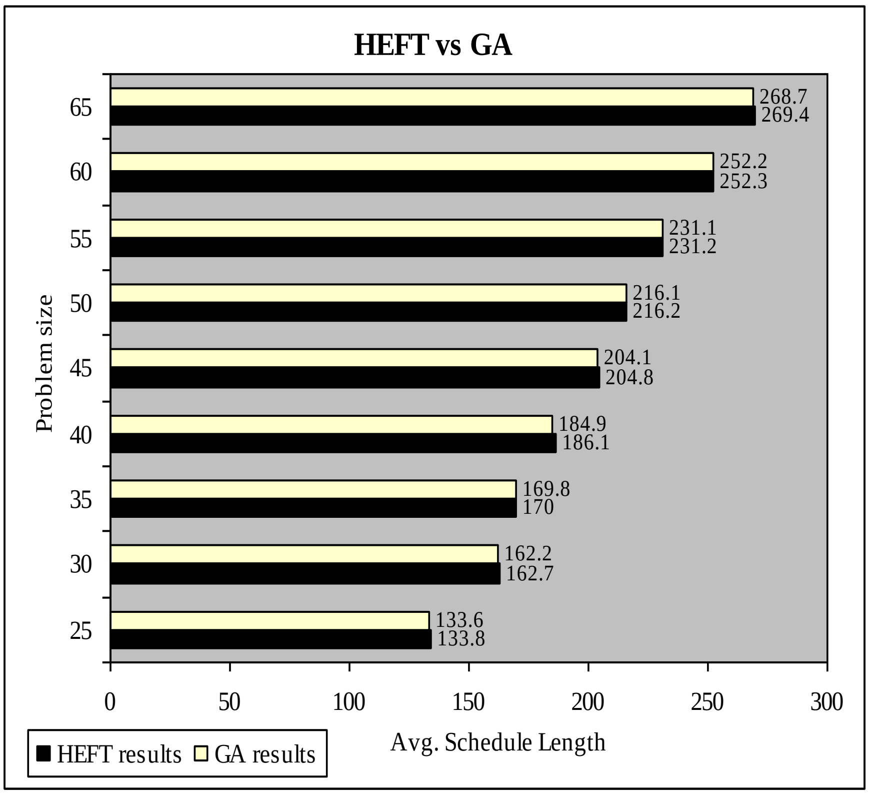


Fig.4.4: HEFT vs. GA Results

4.5.2 Robustness and Guarantee for good solution

During our experiments on GA we noted Average schedule lengths of populations emerging generations after generation (see Fig. 5.12).

Though we have shown results only for problem size 45 in Fig. 5.12 and corresponding data in Table 5.4-(a) to (j), for each problem irrespective of its size we observe that average schedule length is continuously decreasing as more and more generations are evolving even if schedule length corresponding to best chromosome increased in some cases. 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.

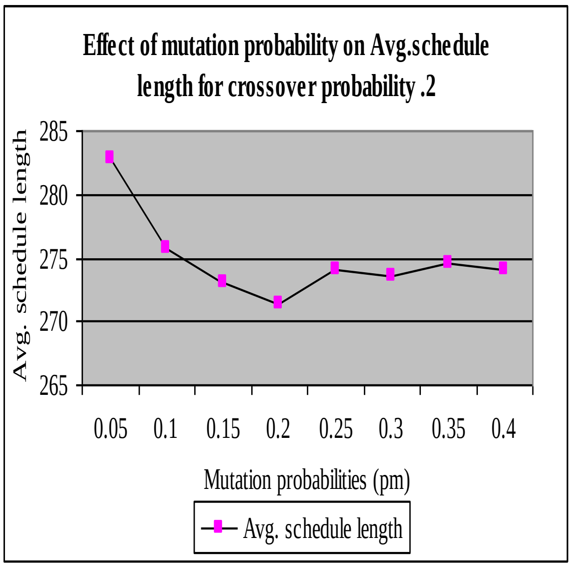
Pc ↓	***Pm*** →						
	0.05	0.1	0.15	0.2	0.25	0.3	0.35
0.2	282.8	275.8	273	271	274.1	273.6	274.6
0.3	282.4	276.7	273.2	272	273.8	273.5	272.6
0.4	284.8	276.7	273.3	273	274.1	275.9	273.8
0.5	280.9	279	274.4	272	273.9	274.3	272.7
0.6	285.2	275.9	272	271	273.4	272.4	273.2
0.7	279.3	275.3	273.7	274	273.2	274	275
Average schedule length	**282.6**	**276.6**	**273.3**	**272**	**273.8**	**274**	**273.7**

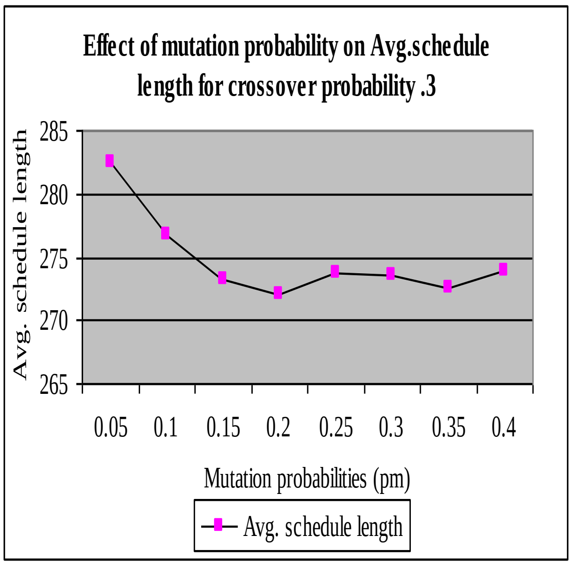
Table 4.5: Average Schedule lengths for problem Size 65.

4.5.3 Effect of mutation probability on performance of Genetic algorithm

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. These results for problem size 65 and crossover probabilities from 0.20 to 0.70 are shown in Fig. 5.13 and corresponding data in Table 5.5. Though results only for problem size 65 are shown here, we observed similar trend in problems of all sizes.

Fig. 5.14 shows 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.





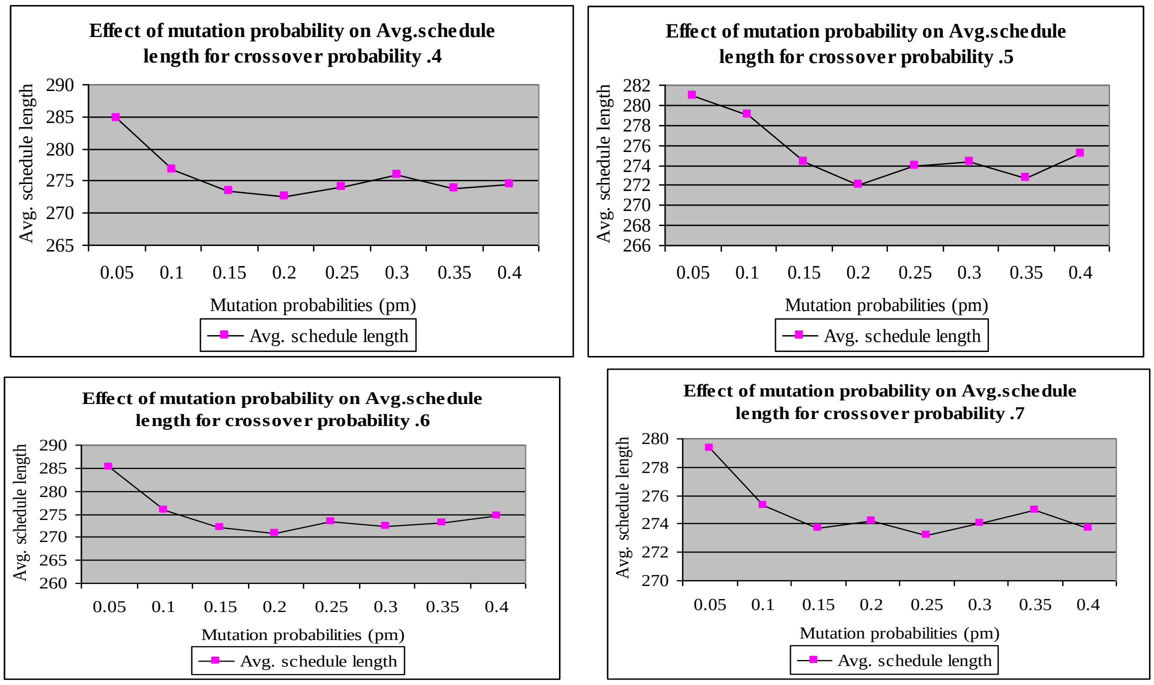


Fig 5.13: Effect of Mutation Probability on Average Schedule Length for Problem Size 65.

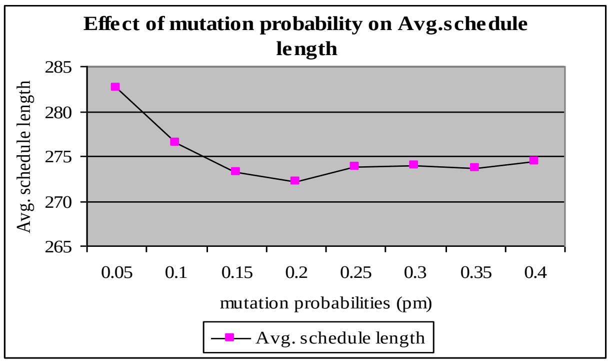


Fig. 5.14: Effect of Mutation Probability on Further Average of Average Schedule Length.

IV. CONCLUSION

A static process scheduling algorithm tries to schedule a set of tasks with known processing and communication characteristics on processors to optimize a performance metric, such as latest completion time for a set of jobs. To avoid solutions involving exhaustive search, researchers have applied heuristics to the problems. However, a heuristic algorithm may work very well for some inputs, but very poorly for others. We would like a scheduling algorithm to be robust, giving good results regardless of the structure of the task it is to schedule. For finding such type of scheduling algorithm, we have developed Genetic Algorithm.

REFERENCES

[1]. A. Chipperfield and P. Flemming, “Parallel Genetic Algorithms”, Parallel and Distributed Computing Handbook, first ed., A.Y. Zomaya, ed., pp. 1,118-1,143.New York: McGraw-Hill, 1996.
[2]. A.Y. Zomaya, C. Ward, and B.S. Macey, “Genetic Algorithms and Scheduling in Parallel Processing Systems:
[3]. Issues and Insight,” Technical Report 97-PCRL-02,Parallel Computing Research Laboratory, Dept. of Electrical and Electronic Eng., The Univ. of Western Australia, 1997.
[4]. Ahmad and Y. Kwok, “On Exploiting Task Duplication inParallel Program Scheduling,” IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 9, pp. 872-892, Sept. 1998.
[5]. Amphlett, R.W. and Bull, D.R., 'Multiprocessor Scheduling for High Quality Digital Audio', IEEE Col.Multiprocessor DSP - Applications, Algorithms and Architectures, London, May 1995, pp. 211-218.

[6]. B. Kruatrachue and T.G. Lewis, “Duplication Scheduling Heuristic, a New Precedence Task Scheduler for Parallel Systems,” Technical Report 87-60-3, Oregon State Univ., 1987.

[7]. B. Malloy, E. Lloyd, and M. Soffa. Scheduling DAG'S for asynchronous multiprocessor execution. IEEE Transactions on Parallel and Distributed Systems, 5(5), May 1994.

[8]. Beasley, D., Bull, D.R. and Martin, R.R., 'An Overview of Genetic Algorithms', University Computing, 1993, Vol. 15, pp. 58-69, 170-181.

[9]. Coffman, E.J., 'Computer and Job-Shop Scheduling Theory', John Wiley & Sons, 1976. Lo, V.M., 'Heuristic Algorithms for Task Assignment in Distributed Systems', IEEE Trans. Computers, Vol. 37, No. 11, Nov. 1988, pp. 1384-97 .

[10]. G.Q. Liu, K.L. Poh, M. Xie, "Iterative list scheduling for heterogeneous computing", Journal of Parallel and Distributed Computing, Vol.65, pp.654-664, 5, 2005.

[11]. Goldberg, D.E., 'Genetic Algorithms in Search, Optimization and Machine Learning', Addison- Wesley Publishing, 1989.

[12]. H. El-Rewini, “Partitioning and Scheduling,” Parallel and Distributed Computing Handbook, A.Y. Zomaya, ed., pp. 239-273. New York: McGraw-Hill, 1996.

[13]. H. El-Rewini, T.G. Lewis, and H.H. Ali, Task Scheduling in Parallel and Distributed Systems. Prentice Hall, 1994.

[14]. H. Topcuoglu, M.Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", IEEE Transactions on Parallel and DistributedSystems, Vol. 13, pp.260-274, 3, 2002.

[15]. Ha, S. and Lee, E.A., 'Quasi-Static Scheduling for Multiprocessor DSP', 1991 IEEE Int. Symposium on Circuits and Systems, Vol. 1, Singapore, June 1991, pp. 352-5.

[16]. Holland, J.H., 'Adaptation in Natural and Artificial Systems', MIT Press, 1975.

[17]. Hou, E.S.H., Hong, R. and Ansari, N.'Efficient Multiprocessor Scheduling Based on Genetic Algorithms', IECON '90, IEEE Industrial Electronics Soc., Pacific Gro. FL., USA, Vol. 2, NOV. 1990, pp. 1239-43.

[18]. Jonathan L. Gross, Jay Yellen, Handbook of Graph Theory, CRC Press.

[19]. Kohler, W.H., 'A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems', IEEE Trans. Computers, Vol. C-24, Dec. 1975,pp. 1235-8.

[20]. M. Srinivas and L.M. Patnaik, “GeneticAlgorithms: A Survey”, Computer, vol. 27, pp. 17-26, 1994.

[21]. Martin Charles Golumbic, Algorithmic Graph Theory and Perfect Graphs: Second Edition, Elsevier, 2004

[22]. P.-C. Wang and W. Korfhage. Process scheduling using genetic algorithms. In IEEE Symp. on Parallel and Dist. Proc., pages 638-641, Texas, USA, Oct. 1995.

[23]. R. Horst and P.M. Pardalos, Handbook of Global Optimization. The Netherlands: Kluwer Academic Publishers, 1995.

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

[25]. S. Darbha, D.P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines", IEEE Transactions on Parallel and Distributed Systems, Vol.9, pp. 87-95, 1, 1998

[26]. T. Tsuchiya, T. Osada, and T. Kikuno, “Genetic-Based Multiprocessor Scheduling Using Task Duplication,” Microprocessors and Microsystems, vol. 22, pp. 197-207,1998.

[27]. 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.