*Int. J. Advance. Soft Comput. Appl., Vol. 2, No. 2, July 2010*
*ISSN 2074-8523; Copyright © ICSRS Publication, 2010*
*www.i-csrs.org*

# A Memetic Algorithm Based Task Scheduling considering Communication Cost on Cluster of Workstations

**S.Padmavathi[1*], S.Mercy Shalinie[2] and R.Abhilaash[3]**

Department of Computer Science & Engineering,
Thiagarajar College of Engineering,
Madurai-625 015, Tamilnadu,
India
[1*]spmcse@tce.edu
[2]shalinie@tce.edu
[3]abhilaash.ravichandran@gmail.com
*Corresponding author

**Abstract**

***Task Scheduling is one of the most challenging problems in parallel and distributed computing. For static scheduling, the program to be parallelized is usually modeled as a Directed Acyclic Graph (DAG). In general, the scheduling of a DAG is a strong NP hard problem. The objective of this problem is minimizing the schedule length considering the communication costs. Genetic algorithm (GA) based technique have been proposed to search optimal solutions from entire solution space. The main shortcoming of this approach is to spend much time doing scheduling and hence, needs exhaustive time. This paper proposes a Memetic Algorithm (MA) to overcome with this shortcoming. Hill Climbing algorithm as local search is applied in the proposed memetic algorithm. Extended simulation results demonstrate that the proposed method outperforms the existing GA-based method, producing the optimal schedule.***

**Keywords**: *Direct Acyclic Graph (DAG), Task scheduling, Genetic algorithm (GA), Memetic Algorithm (MA), Hill Climbing Algorithm, Local search, schedule length.*

## 1 Introduction

Performance of the program critically depends on the partitioning of the program and the scheduling of the resulting tasks onto the physical processors. Classical scheduling model assumes that each task is processed on one processor at a time. The objective of the task scheduling problem is to minimize the makespan (schedule length) i.e., the overall computation time of any application represented as Directed Acyclic Graph (DAG). Optimal scheduling of tasks of a DAG onto a set of processors is a strong NP-Hard problem. It has been proven to be NP-Complete for which optimal solutions can be found only after an exhaustive search. The optimal solutions for many scheduling heuristics stated in the literature [1, 2] have been proposed in the past. Early scheduling algorithms did not take communication into account. But due to the increasing importance for parallel performance, the consideration of communication was included in the proposed approach. The consideration of communication cost [3] is significant to produce an accurate and efficient schedule length.

List scheduling algorithms such as Heterogeneous Earliest Finish Time (HEFT), Critical Path On a Processor (CPOP) [4] and Performance Effective Task Scheduling (PETS) [5] are complex in nature and take higher complexity in implementation. The HEFT algorithm uses a recursive procedure to compute the rank of a task by traversing the graph upwards from the exit task and vice-versa for CPOP. The rank of a task is the length of the critical path from the exit task to that task. The rank computation is a recursive procedure and also complex in nature for both the algorithms. In [6, 20] a Simple Genetic Algorithm (GA) for multiprocessor task scheduling is proposed.

Some GA parameters are to be used for mapping and scheduling general task graph [7] whereas in [8] bichromosomal representation for task scheduling problem is used. GA's [9-12] are a class of random search techniques for the task scheduling problem. Although GA provide good quality schedules, their execution times are significantly higher than other alternatives. Extensive tests are required to find optimal values for the set of control parameters used in GA – based solutions [13].

GA for static scheduling of *m* tasks to *n* processors based on k-way partitioning was developed in [14]. Successive improvements to the initial schedule were made through reproduction, mutation and one-point crossover operators. The traditional methods such as Branch and Bound, Divide and Conquer and Dynamic programming give the global optimum, but it is time consuming [15]. The researchers [16] have derived optimal task assignments to minimize the sum of the task execution and communication costs with the Branch and Bound method and evaluated the computational complexity of this method using simulation techniques. Modern heuristic techniques [17] are general purpose optimization algorithms. Their efficiency or applicability is not tied to any specific problem-domain. To improve the efficiency of the heuristic based approach, there exist guided random search techniques such as Simulated Annealing, Tabu Search, Particle Swarm Optimization, Genetic Algorithm etc.

The GA is not well suited for fine-tuning structures which are close to optimal solution [18]. Memetic Algorithm (MA) are evolutionary algorithms (EAs) that apply a separate local search process to refine individual (i.e. improve their fitness by Hill-Climbing) [22, 23]. They are a special kind of GA with local search. The local search may be Hill Climbing or Tabu search or Simulated Annealing [32]. The memetic algorithms [19] can be viewed as a marriage between a population-based global technique and a local search made by each of the individuals. Like GA, MA is also a population-based approach. A GA with local search, which is known as Memetic Algorithm (MA), can use one or more local search techniques [23]. Here, the Hill Climbing algorithm is used as a local search algorithm. They have shown that they are orders of magnitude faster than traditional GAs for some problem domains [21]. MA yields faster convergence when compared to GA, because the balance between the exploration and exploitation is in the search process [30, 31].

MA is the subject of intense scientific research and has been applied to a multitude of real world problems [24]. It represents one of the recent emerging areas of research in evolutionary computation. The term MA is now widely used as synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search [24]. Quite often, MA is also referred to in the literature as Baldwinian EAs, Lamarckian EAs, Cultural algorithms or Genetic Local Search or hybrid genetic algorithm [29]. In case of hybrid flow shop scheduling problem, MA produces better quality solution and it is efficient when compared to GA and constraint programming based branch and bound algorithm [26].

To validate the performance of the proposed approach, highly communicating task graph like Gaussian elimination is generated and also tested with randomly generated DAGs.

The paper is organized as follows: introduction is followed by the problem definition which is presented in Section 2. Section 3 discusses the fundamentals of MA. Section 4 introduces the proposed algorithms and implementation aspects. Section 5 presents the experimental results and discussions. Finally the conclusion and future research direction are presented in Section 6.

## 2 Problem Definition and Background

An application program is represented by a Directed Acyclic Graph (DAG).A DAG is a directed acyclic graph $G =(V,E,w,c)$ representing a program $P$. Here, $V$ is a set of task nodes and $E$ is a set of communication edges, corresponding to the dependency among tasks. An edge $e_{ij} \in E$ represents the communication from node $n_i$ to node $n_j$. The positive weight $w(n)$ associated with node $n \in V$ represents its computation cost and the nonnegative weight $c(e_{ij})$ associated with edge $e_{ij}$ $\in$

*E* represents its communication cost. The communication cost between two nodes assigned to the same processor is assumed to be zero. '<' represents a partial order on *V*. For any two tasks $n_i$, $n_k$ the existence of the partial order $n_i < n_k$ means that $n_k$ cannot be scheduled until task $n_i$ has been completed, hence $n_i$ is a predecessor of $n_k$ and $n_k$ is a successor of $n_i$. The task executions of a given application are assumed to be non-preemptive. In a given task graph, a task without any predecessor is called an *entry task* and task without any child is called an *exit task*. Here, it is assumed that there is one entry and exit task in DAG.

A node cannot begin execution until all its inputs have arrived and no output is available until the computation has finished. It can be defined as precedence constraints which is as follows

$$t_s(n_j, P) = t_f(e_{ij}) \qquad (1)$$

where $t_s\,(n_j,P)$ denotes the start time of the node in the processor "*P*" and $t_f\,(e_{ij})$ is the edge finish time of the communication associated with $e_{ij}$. Data Ready Time (DRT) $t_{dr}$ of a node can be calculated as follows:

$$t_{dr}(n_j, P) = \max\{t_f(e_{ij}) + w(e_{ji})\} \qquad (2)$$

and hence for a valid schedule,

$$t_s(n, P) \geq t_{dr}(n, P) \qquad (3)$$

A task graph for Gaussian elimination for 3 x 3 matrix is shown in Fig.1 and its computation cost matrix is shown in Table 1. Let *EST* ($n_i$, $p_j$) and *EFT* ($n_i$,$p_j$) are the Earliest Start Time and Earliest Finish Time of task $n_i$ on $p_j$, respectively. For the entry task $v_{\text{entry}}$, $EST(v_{entry}, p_j) = 0$, and for the other tasks in the graph, the EST and EFT values are computed recursively, starting from the entry task, as shown in Eqns. (4) and (5). In order to compute the EFT of a task $n_i$, all immediate predecessor tasks of $n_i$ must have been scheduled.

$$EST(n_i, p_j) = \max\{avail[j], \max(AFT(n_k + C_{i,k}))\} \qquad (4)$$
$$where\ n_k \in pred(n_i)$$

$$EFT\ (n_i, p_j) = w_{ij} + EST\ (n_i, p_j) \qquad (5)$$

where $pred(n_i)$ is the set of immediate predecessor tasks of task $n_i$ and *avail[j]* is the earliest time at which processor $p_j$ is ready for task execution. If $n_k$ is the last assigned task on processor $p_j$, then *avail[j]* is the time that processor $p_j$ completed the execution of the task $n_k$ and it is ready to execute another task. The inner max block in the EST equation returns the ready time, i.e., the time when all the data needed by $n_i$ has arrived at processor $p_j$. After a task $n_k$ is scheduled on a processor $p_j$, the earliest start time and the earliest finish time of $n_i$ on processor $p_j$ is equal to the Actual Start Time $AST(n_k)$ and the Actual Finish Time $AFT(n_k)$ of task $n_k$, respectively. After all tasks in a graph are scheduled, the schedule length (i.e the overall completion time) will be the actual finish time of the exit task, $n_{exit}$..

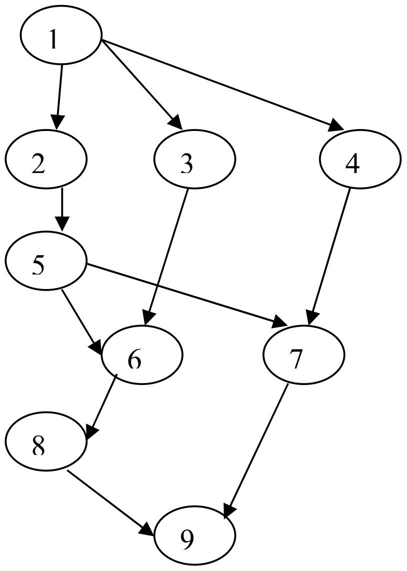

Fig. 1. Gaussian elimination task graph represented by DAG for 3 x 3 matrix

Table 1: Computational cost matrix (W) for Fig.1

| *Task* | *P1* | *P2* | *P3* |
|---|---|---|---|
| 1 | *3* | *3* | *3* |
| 2 | *4* | *5* | *4* |
| 3 | *4* | *6* | *4* |
| 4 | *5* | *3* | *5* |
| *5* | *3* | *7* | *2* |
| 6 | *3* | *6* | *1* |
| 7 | *5* | *3* | *6* |
| 8 | *2* | *4* | *5* |
| 9 | *5* | *8* | *5* |

Priority is computed and assigned to each task based on the following attributes namely, Average Computation Cost (ACC), Data Transfer Cost (DTC) and Rank of the Predecessor task (RPT).The ACC of a task is the average computation cost on all the '*m'* processors and it is computed by using Eqn. (6).

$$ACC(v_i) = \sum_{j=1}^{m} \frac{w_{i,j}}{m} \tag{6}$$

The DTC of a task $v_i$ is the amount of communication cost incurred to transfer the data from task $v_i$ to all its immediate successor task and it is computed at each *level l* using Eqn. (7)

$$DTC(v_i) = \sum_{j=1}^{n} C_{i,j} : i < j \tag{7}$$

Where $n$ is the number of nodes in the next level.

*DTC = 0* for exit tasks

The RPT of a task $v_i$ is the highest rank of all its immediate predecessor task and it is computed using Eqn. (8)

$$RPT(v_i) = Max\{rank(v_1), rank(v_2), ...rank(v_h)\} \tag{8}$$

Where $v_1, v_2, v_3 ........ v_h$ are the immediate predecessor of $v_i$

*RPT = 0* for entry task

Rank is computed for each task $v_i$ based on its ACC, DTC, RPT values. Here, the maximum rank of predecessor tasks of task $v_i$ as one of the parameters to calculate the rank of the task $v_i$ and the rank computation is given by Eqn. (9).

$$rank(v_i) = round\{ACC(v_i) + DTC(v_i) + RPT(v_i)\} \tag{9}$$

Priority is assigned to all the tasks at each *level l*, based on its rank value. At each level, the task with highest rank value receives the highest priority followed by task with next highest rank value and so on. Tie, if any, is broken using ACC value. The task with minimum ACC value receives the higher priority.

Finally the objective function *f(x)* can be defined as

$$f(x) = \min(schedule\ \ length) \tag{10}$$

Where schedule length is defined as

$$Schedule\ Length = \max\{AFT(n_{exit})\} \quad (11)$$

## 3 Memetic Algorithm

Memetic Algorithm (MA) combine GA with local search. MA are inspired by memes (Dawkins, 1976), pieces of mental idea like stories, ideas and gossip, which reproduce (propagate) themselves through population of memes carriers. Corresponding to the selfish gene idea (Dawkins, 1976) in this mechanism each meme uses the host (the individual) to propagate itself further through the population, and in this way the population competes with different memes for the limited resources.

MA starts with several alternative solutions to the optimization problem, which are considered as individuals in a population. These solutions are coded as binary strings called chromosomes. Suitable encoding plays an important role in deciding the performance of MA. The population is initialized at random or using a heuristic. To form a new population for the next generation, higher quality individuals are selected. The selection phase is identical in form to that used in the classical GA selection phase. Local search is performed to select the best chromosome from the pool of available chromosomes. Once the best chromosome has been selected, they are subjected to crossover and mutation to generate new individuals. Finally, one best chromosome is selected by applying the final local search. The role of local search in MA is to search and locate the local optimum more efficiently than the GA. Fig.3 explains the generic implementation of Memetic Algorithm.

```
1. Encode solution space
2. (a) set pop_size, max_gen, gen=0;
   (b) set cross_rate, mutate_rate;
3. initialize population
4. while(gen < gensize)
Apply generic GA
Apply local search
end while
Apply final local search to best chromosome
```

Fig.3. The Memetic Algorithm

### 3.1 Hill Climbing local search algorithm

Local search can be thought of as the process of an individual improving its idea of the solution. The Hill Climbing search algorithm is a local search algorithm and is shown in Fig. 4. It is nature-based stochastic computational technique [25]. It is used to execute local search to find better solutions in the neighborhood of the current solution produced by GA in each iterations. When the termination condition is met, it returns with the best solution.

```
Best solution ←initial solution
While (termination condition is not satisfied) do
New solution ← neighbors (solution after cross over and mutation)
If New solution is better than best solution then
Best solution ← New solution
End if
End while
```

Fig .4**.** The Hill Climbing local search algorithm

## 4 The Proposed Method

### 4.1 Encoding

The generic formulation of a problem begins with the definition of an appropriate chromosome encoding. Each chromosome encodes a schedule solution. In order to achieve good performance, the chromosome should be simple, because this permits one to employ simple and fast operators.

For task-scheduling, a chromosome represents a solution to the scheduling problem; in other words a schedule. A schedule consists of the processor allocation and the start time of each node of the task graph. The representation of the chromosome holds the information that serves as an input for a heuristic search to create a schedule. There are three basic elements to choose among. The first is the list of tasks to be scheduled. The second is the order in which these tasks should be executed on a given processor and the third is the list of processors to which these tasks should be assigned.

Each chromosome is represented as a group of genes i.e. task-processor pair $(T_i, P_i)$ indicating that task $T_i$ is assigned to the processor $P_i$ shown in Fig. 5. The position of genes in a chromosome represents the order in which the tasks should be executed. For example the following chromosomal representation show that task 1 and task 2 should be executed on processor 1 and task 3 on processor 2. It also indicates that task 2 is executed first followed by task 3 and followed by task 1.

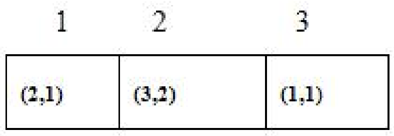

Fig. 5.Chromosomal representation

## 4.2 Initial population

Most of the scheduling heuristics generate the initial population randomly with the necessary care on feasible solutions. The population is created randomly i.e. a predefined number of chromosomes are generated, the collection of which form the initial population. Here the initial population is generated based on the priority calculation of the tasks at each level as shown in Table 2.

Table 2: The DTC, ACC, RPT, Rank and Priority values for the tasks in Fig.1

| *Level* | Task | ACC | RPT | DTC | Rank | Priority |
|---|---|---|---|---|---|---|
| *1* | *1* | *3* | *0* | *6* | *9* | *1* |
| *2* | *2* | *4.33* | *9* | *4* | *17* | *3* |
| *2* | *3* | *4.67* | *9* | *6* | *19* | *2* |
| *2* | *4* | *4.33* | *9* | *7* | *20* | *1* |
| *3* | *5* | *4.0* | *17* | *17* | *38* | *1* |
| *4* | *6* | *3.33* | *38* | *10* | *51* | *2* |
| *4* | *7* | *4.67* | *38* | *11* | *53* | *1* |
| *5* | *8* | *3.67* | *51* | *12* | *66* | *1* |
| *6* | *9* | *6* | *66* | *0* | *72* | *1* |

### Fitness function

As the objective of the task scheduling problem is to find the shortest possible schedule, the fitness of a chromosome is directly related to the length of the associated schedule. Here the fitness value is determined by the earliest finish time of the last task.

## 4.3 Selection

In this step, the chromosomes in the population are ranked first based on their fitness value from the best to the worst. The chromosomes with least fitness values are ranked as best chromosomes. This process of obtaining the best chromosome is called as selection. This is done using local search from the pool of available chromosomes.

## 4.4 Reproduction

Reproduction process forms a new population of chromosomes by selecting chromosome in the old population based on their fitness value through Crossover and mutation.

**Cross over**

The cross over operator is the most significant one since it implements the principle of evolution. New chromosomes are created with this operator by combining two selected parent chromosomes and swaps second part of each chromosome after a randomly selected point. This is equivalent to assigning a subset of tasks to different processors. Single point and two point crossovers are alternatively performed and the crossover probability is selected randomly.

**Mutation**

This operator is applied with a lower probability (about 0.1 or less) than the crossover operator. Its main purpose is to serve as a safeguard to avoid the convergence of the state search to a locally best solution. Here the partial-gene mutation is employed. It takes each chromosome from the fittest ones and changes a randomly selected gene $(T_i, P_i)$ to $(T_i, P_j)$ which introduces diversity each time it is applied, and consequently the population continues slowly to improve. Therefore the probability of crossover and partial-gene mutation is not fixed in the proposed algorithm.

## 4.5 Local Search

The Hill climbing search algorithm is a local search algorithm that iteratively performs a neighborhood search to pick best chromosome from a pool of available chromosomes. When the termination criterion is met, the search algorithm terminates and returns the best solution. It is explained in Fig. 4.

## 4.6 Termination Criteria

When no improvement solution has been found over the last *n* iterations, the algorithm terminates. Typically this value lies between 50 to 500 depending on the desired quality of the solution and the size of the problem. Since for a larger problem, improvement moves are likely to be found with lower frequency.

The proposed memetic algorithm is as follows:

```
1. Generate the initial population of size M based on task priority at each level
   and calculate the fitness value of each Chromosome based on earliest finish
   time of the last task.
2. Select the fittest chromosome from the initial pool based on the Least Fitness
   Value (schedule length) using local search.
3. Perform Crossover and Partial gene mutation with varying probabilities.
4. Evaluating the chromosomes obtained from the step 3 and form a pool of fittest
   Chromosomes using local search.
5. Repeat steps 3 and 4 until termination criteria is met.
```

Fig.6. The proposed memetic algorithm

## 5 Results and Discussions

In this section a number of experiments are carried out which outlines the effectiveness of the proposed algorithm. The purpose of these experiments is to compare the performance of memetic algorithm approach with genetic algorithm approach for the task scheduling problem. Although the memetic algorithm is a GA combined with the Hill Climbing algorithm as a local search, it is not necessarily the case that the genetic parameters are the most ideal for a memetic algorithm. The experiments were tested on a cluster of workstations consisting of 32 node HP Proliant cluster.

DAGs are generated randomly with different communication cost whose size varies from 10 to 50. Highly communication intensive application like Gaussian Elimination task graph is also generated with matrix size varying from 3 to 15. The results are compared for varying population size, where the size ranges from 5 to 200. The tasks are selected for an initial pool according to the priority value as shown in Table 2 for the Gaussian Elimination task graph in Fig.1. Then they are selected according to their fitness value.

For the proposed approach, the effects of the different population size and different number of iterations are investigated and the results are depicted in Figs. 7 and 8. The performance of MA improves when the population size is increased.

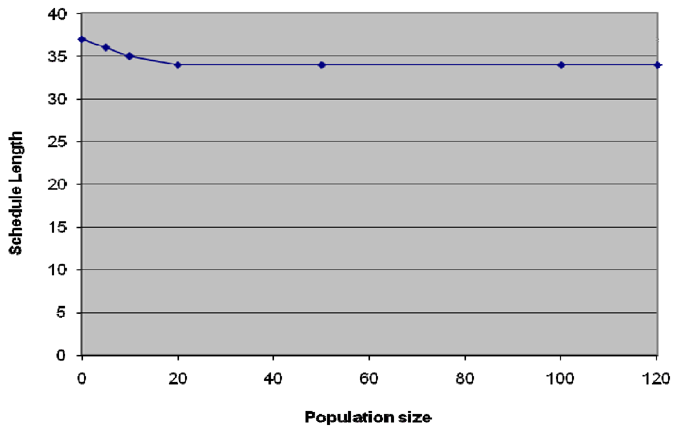

Fig. 7. Schedule length Vs Population size for Gaussian Elimination Task Graph

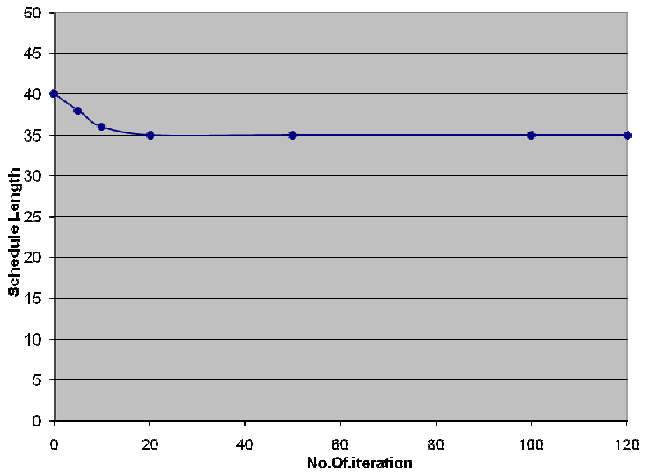

Fig. 8.Schedule length Vs No.of.Iteration for Gaussian Elimination task Graph

MA converges very fast when compared to GA as shown in Fig. 9. The results of both MA and GA are compared by varying the number of iterations from 5 to 250 for the Gaussian Elimiation task graph.Traditionally GAs suffer from slow convergence to locate a precise enough solution because of their failure to exploit local information. But MAs are hybrid GAs that combine global and local search which uses GA, to perform exploration while the local search method performs exploitation.

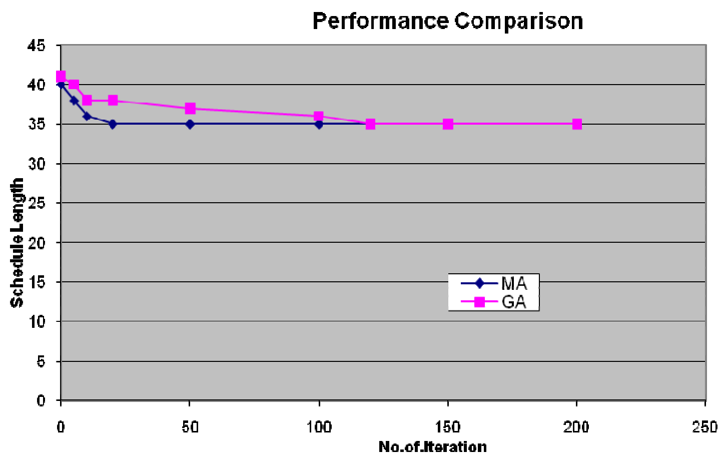

Fig .9 Performance Comparison of GA Vs MA

MA is compared with the classical list scheduling algorithms like HEFT, CPOP and PETS algorithm for the Gaussian Elimination task graph whose matrix size is 5 x 5. The performance of MA is compared with existing HEFT, CPOP and PETS algorithm is shown in Fig.10. From the above results, the proposed MA performs well when compared to other list scheduling algorithms. Since the three algorithms are based on list scheduling and the method producing the scheduling list and the priority assigning rules are different.

The HEFT algorithm uses a recursive procedure to compute the rank of a task by traversing the graph upwards from the exit task. Based on the rank, priority is assigned to each task. The CPOP algorithm uses a reverse fashion of calculating the rank by traversing the graph from downwards from the entry task. The PETS algorithm calculates rank based on ACC, DTC and RPT values [4, 5]. But in the proposed MA, first chromosomes are encoded as task-processor pair or as a schedule solution. Task priorities are calculated like PETS algorithm. Then the processors are assigned to each task pseudo-randomly. The chromosomes are encoded to represent the task-processor pairs. The best chromosome is selected using local search. The fitness value of that best chromosome gives the schedule length.

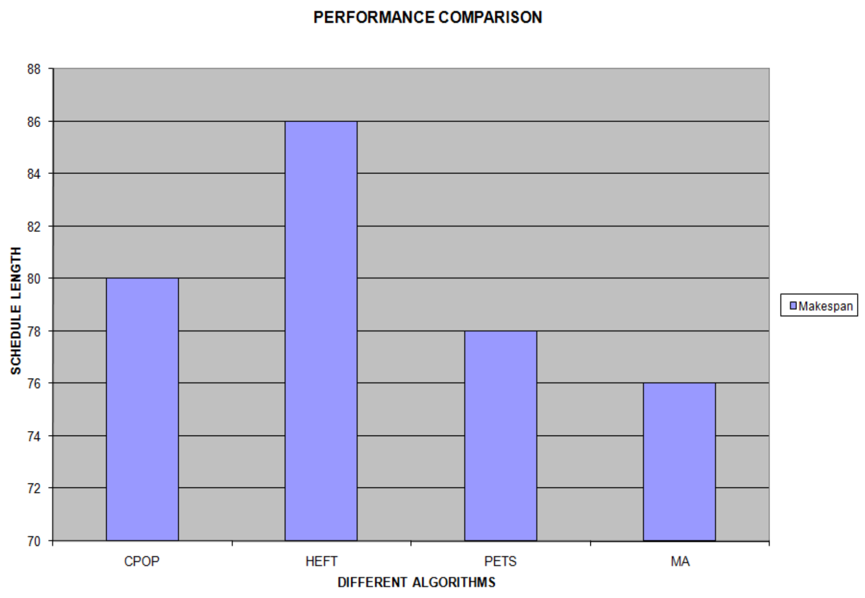

Fig. 10.Performance Comparison

## 6 Conclusion

The proposed MA is appropriate for scheduling DAG structured applications onto homogeneous computing system with different topologies. However GA's and MA's are gaining popularity due to their effectiveness of solving the optimization problems within a reasonable time. Experimental results showed that the proposed approach is better than GA in almost all cases. MA converges very fast when compared to GA, hence the proposed approach outperforms all the existing heuristics for the task scheduling problem. The future enhancement of this work is to introduce contention awareness in task scheduling using MA.

**ACKNOWLEDGEMENTS**

The authors owe their acknowledgement to the Management and Principal of Thiagarajar College of Engineering for their encouragement and support.

## References

[1] A.Gerasoulis and T.Yang, "A Comparison of clustering heuristics for scheduling DAGs on multiprocessors", *Journal of Parallel and Distributed Computing*, Vol.16, No.4, (1992), pp.276-291.

[2] Y.kwok and I.Ahmad, “Benchmarking the task graph scheduling algorithms”, *In Proc.Int.Par.Processing Symposium on Parallel and Distributed Processing(IPPS/SPDP-98)* ,USA, Florida,(1998),pp.531-537.

[3] B.S. Macey and A.Y.Zomaya, “A Performance evaluation of CP list scheduling heuristics for communicating Intensive task graphs”, *In Parallel Processing Symposium*,(1998),pp.538-541.

[4] Topcuoglu,H.,S.Hariri and M.Y.Wu, “Performance Effective and Low Complexity Task Scheduling Algorithm scheduling for heterogeneous computing “, *IEEE Transaction on Parallel and Distributed Systems*,Vol.13,No.3,(2002).

[5] E.Illavarasan and P.Thambidurai, “Low complexity performance effective task scheduling algorithm for Heterogeneous computing environments”, *Journal of Computer sciences*, Vol.3, No.2, (2007), pp.94-103.

[6] E.S.Hou,N.Ansari and H.Ren, “A Genetic algorithm for Multiprocessor Scheduling”,*IEEE Transaction on Parallel and Distributed Systems*, Vol.5, No.2, (1994).

[7] Harmel Singh,Ardou Youssef, “Mapping and Scheduling heterogeneous Task Graphs using Genetic algorithms”.

[8] Michael Rinchart,Vidakiazod and Shurva S.Bhattachariya, “A Modular Genetic algorithm for scheduling task graphs”,*Technical report UMIACS-TR 2003-66*, Institute of Advanced Computer Studies,University of Maryland at College park,June 2003.

[9] L. Wang, H. J. Siegel, V. P. Rowchoudhry and A. A. Maciejewski, “Task matching and scheduling in heterogeneous computing environments using a genetic algorithm-based approach”, *Journal of Parallel and Distributed Computing*, Vol. 4, (1997), pp.8-22.

[10] M. K. Dhodhi, I. Ahmad and A. Yatama, “An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems”, *Journal of Parallel and Distributed Computing*, Vol. 62, (2002),pp.1338-1361.

[11] S. C. Kim and S. Lee, “Push-pull: Guided search DAG scheduling for Heterogeneous clusters”, *Proc. Intl. Conf. Parallel Processing*, 2005.

[12] S. W. Annie, H. Yu, S. Jin and K. C. Lin, “An incremental genetic algorithm approach to multiprocessor scheduling”, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, (2004),pp.824-834.

[13] T. D. Braun, H. J. Siegel, N. Beck and L. L. Boloni, “A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems”, *Proc. 8th Workshop on Heterogeneous Processing*”, (1999),pp.15-29.

[14] S.M.El-Gendy, "Task Allocation using Genetic algorithms", *MS Thesis, University* of Louisville,(1994).

[15] Tzu-Chiang Chiang, Po-Yin Chang and Yueh-Min Huang, "Multi-Processor Tasks with Resource and Timing Constraints Using Particle Swarm Optimization", *IJCSNS International Journal of Computer Science and Network security*, Vol.6, No.4 (2006), pp.71-77.

[16] Dar-Tzen Peng,Kang G.Shin and Tarek F.Abdelzaher, "Assignemnt and Scheduling Communicating Periodic tasks in Distributed Real-Time Systems", *IEEE Transaction on Software Engineering,* Vol.23, No.12, (1997), pp.745-758.

[17] Abdelmaged Elsadek A and EARL Wells, "A heuristic model for Task allocation in heterogeneous distributed computing systems",*The International Journal of Computers and Their Applications*, Vol.6, No.1, (1999), pp.1-36.

[18] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, (1989).

[19] P, Moscato, "on evolution, scorch, optimization. Genetic algorithms and Martial arts: toward memetic algorithms", *Technical report*, California, (1989).

[20] D.Liu,Y.Li and M.Yu, "A Genetic algorithm for Task Scheduling in Network Computing Environment ", *Fifth International Conference on Algorithms & Architecture for Parallel Processing, ICA3PP'02*.

[21] Poonam Garg, "A Comparison between Memetic Algorithm and Genetic Algorithm for the Cryptanalysis of Simplified Data Encryption Standard algorithm", *International Journal of Network Security & Its Applications(INJSA)*,Vol. 1,No 1,(April 2009).

[22] Shawki Areibi , Medhat Moussa and Hussein Abdullah , " A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques", Research Paper.

[23] S.M.Kamrul Hasan et al, "Memetic algorithm for solving job-shop scheduling problems", *Journal of Memetic Computing*, Vol.1, (2009).

[24] http:\\en.wikipedia.org\wiki\Memetic_algorithm.

[25] Junying Chen et al, "Particle Swarm Optimization with Local Search", *IEEE proceedings,* (2005).

[26] Antonie Jouglet , Ceyda Oguz and Marc Servaux, " Hybrid Flow-Shop: a Memetic algorithm using Constraint-Based Scheduling for Efficient Search", *Journal of Mathematical modelling and Algorithm, Springer Netherlands*,(2009),pp.271-292.

[27] Mohammad Sadegh et al, “A Novel Method for task scheduling in Distributed Systems Using Memetic Algorithm”, *IEEE Proceedings of International Conference on Communication, Theory, Reliability and Quality of Service, CTRQ*,(2009), pp:58-62.

[28] Jalil Layegh,Fariborz Jolai,Mohsen Sadegh Amalnik, “A Memetic algorithm for minimizing total weighted completion time on a single machine under step-deterioration”, *Advances in Engineering Software, Elsevier,* Vol.40, No.10,(October 2009), pp. 1074-1077.

[29] Quang Huy Nguyen et al, “A probabilistic Memetic Framework”, *IEEE Transaction on Evolutionary Computation*, Vol.13, No.3,(2009),pp.604-623.

[30] Fengjie Wu, “A Framework for Memetic Algorithm”, *Thesis of University of Auckland*, (2001).

[31] Y.S.Ong et al, “Classification of Adaptive MA: A Comparative Study”, Technical Report, (2005).

[32] J.Digalakis and K.Margaritis, “Performance Comparison of Memetic Algorithm”,*Complexity International Journal*,Vol.10, (2005).