CHAPTER 5

MEMETIC ALGORITHM BASED TASK SCHEDULING

5.1 INTRODUCTION

Metaheuristics approaches have shown their effectiveness for a wide variety of hard optimization problems and also for multi-objective optimization problems. Their efficiency or applicability is not tied to any specific problem-domain. After many years of research, a hybrid method capable of providing a better solution in less time is achieved through the combination of GA and other Evolutionary Algorithms with the optimization methods. One form of such hybrid methods is Memetic Algorithms (MA). The key feature of the MA is to use the available knowledge of the problem, such as the approximation algorithms and local search techniques, to solve the problem (Corne et al 1999). The added heuristic is the main difference between the GA and the MA which makes the MA one of the most successful approximation techniques for Non-Polynomial (NP) optimization problems (Digalakis and Margaritis 2005). However, premature convergence is an inherent characteristic of such classical Genetic Algorithms which makes them incapable of searching numerous solutions of the problem domain. The MA uses a local search technique to reduce the likelihood of premature convergence.

By hybridizing the population based evolutionary searching ability of the GA with local improvement abilities of Hill Climbing (HC) and Tabu Search (TS), effective approaches have been proposed for Task Scheduling
with an objective to minimize the makespan in this research work. The performance of the proposed metaheuristic is more optimal than reported in the literature.

5.2 GENERIC GENETIC ALGORITHM

The Genetic Algorithm (GA) is a stochastic global search and optimization method that mimics the metaphor of natural biological evolution. The GA is based upon the Darwinian evolution theory. It is modeled on a relatively simple interpretation of the evolutionary process; however, it has proven to be a reliable and powerful optimization technique in a wide variety of applications. Holland in 1975 was the first to propose the use of genetic algorithms for problem solving. Goldberg (1989) was also a pioneer in the area of applying genetic processes to optimization. As an optimization technique, the Genetic Algorithm simultaneously examines and manipulates a set of possible solution. Over the past few years numerous applications and adaptations of GA have appeared in the literature (Kafil and Ahmed 1998).

The GA maintains a population of individuals that represent candidate solutions. Each individual is evaluated to give some measure of its fitness to the problem from the objective function. In each generation, a new population is formed by selecting the more fit individuals based on a particular selection strategy. Some members of the new population undergo genetic operations to form new solutions. The two commonly used operations are crossover and mutation. Crossover is a mixing operator that combines genetic material from selected parents. Mutation acts as a background operator and is used to search the unexplored search space by randomly changing the values at one or more positions of the selected chromosome.

5.2.1 Genetic Algorithm Approach

During each iteration of the algorithm, the processes of selection, reproduction and mutation take place in order to produce the next generation
of solutions. The Genetic Algorithm begins with a randomly selected population of chromosomes represented by strings. The GA uses the current population of strings to create a new population so designed that the strings in the new generation are on average better than those in current population (the selection depends on their fitness value). The selection process determines as to which string in the current will be used to create the next generation. The crossover process determines the actual form of the string in the next generation. Here two of the selected parents are paired. A fixed small mutation probability is set at the start of the algorithm. The crossover and mutation processes ensure that the GA can explore new features that may not be in the population yet. It makes the entire search space reachable, despite the finite population size.

5.2.2 Working Principle of Genetic Algorithm

Starting with an initial population, the genetic algorithm exploits the information contained in the present population and explores new individuals by generating offspring using the three genetic operators namely, reproduction, crossover and mutation, which can then replace the members of the old generation. Fitter chromosomes have higher probabilities of being selected for the next generation. After several generations, the algorithm converges to the best chromosomes, which hopefully represent the optimum or near optimal solution. The above process is pictorially represented in Figure 5.1.

Problem Representation

The choice of how to encode the solution on a chromosome is of primary importance to the success of the genetic algorithm approach to a problem. The encoding of information on the chromosomes should be right for the problem rather than specific to the problem. The encoding should be able to represent all of the truly relevant parameters of the problem and
should avoid other parameters. Using parameters that aren’t directly relevant will cause the genetic algorithm to be subject to changes in the problem that would not otherwise affect it, thereby making it no more useful than a specialized heuristic. Care must be taken to ensure that the selected encoding can uniformly represent all the possible solutions to the problem, regardless of any prior expectation of their viability.

Population Initialization

There are two parameters that have to be decided for initialization namely the initial population size and the procedure to initialize the population. Initially researchers thought that the population size needed to increase exponentially with the length of the chromosomes string in order to generate good solutions. Recent studies have shown that satisfactory results can be obtained with a much smaller population size. There are two ways to generate the initial population namely random initialization and heuristic initialization. The initial chromosomes need not represent a legal solution.

Fitness Function

GAs mimics the survival of the fittest principle of nature to make the search process. Therefore, they are naturally suitable for solving the maximization problems. Minimization problems are usually transformed into maximization problems by suitable transformation. In general, a fitness function \( F(x) \) is first derived from the objective function and used in successive genetic operations. Certain genetic operations require that the fitness function be non-negative, although certain operators do not have this requirement. For Maximization problems, the fitness function can be considered to be the same as the objective function and hence \( F(x) = f(x) \). For minimization problem, the given function, \( f(x) \) is transformed using Equation (5.1) which is often used.
This transformation does not alter the location of the minimum but converts a minimization problem to an equivalent maximization problem. The fitness function value of a string is known as the string’s fitness.

\[
F(x) = \frac{1}{1 + f(x)}
\]  

(5.1)

Figure 5.1 Flowchart of Genetic Algorithm
5.3 MEMETIC ALGORITHM (MA)

Memetic Algorithms (MA) represent one of the recent growing areas of research in evolutionary computation. Memetic Algorithms (MAs) are Evolutionary Algorithms (EAs) that apply a separate local search process to refine individuals (i.e. improve their fitness by Hill Climbing). These methods are inspired by the models of adaptation in natural systems that combine evolutionary adaptation of populations of individuals with individual learning within a lifetime.

MAs constitute an extremely powerful tool for tackling combinatorial optimization problems. Traditional NP Optimization problems constitute one of the most typical battlefields of MAs, and a remarkable history of successes has been reported with respect to the application of MAs to such problems. Other such application areas of MAs include machine learning, robotics, engineering, electronics, bioinformatics, oceanography, and many more.

The Memetic Algorithm (MA) combines the GA with the local search. The MAs are inspired by memes (Dawkins 1976), pieces of mental idea like stories, ideas and gossip, which reproduce (propagate) themselves through the 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. Quite often, MA is also referred to in the literature as Baldwinian EAs, Lamarckian EAs, cultural algorithms or genetic local search.
5.3.1 Comparisons between GA and MA

MAs are population based heuristic search approaches for optimization problems similar to Genetic Algorithms (GAs). GAs, however, rely on the concept of biological evolution, but MAs, in contrast, mimic cultural evolution. While in nature, genes are usually not modified during an individual’s lifetime, memes are. They are the evolutionary algorithms that include a stage of individual optimization or learning as a part of their search strategy.

A population-based search algorithm called Genetic Algorithm (GA) is commonly used to solve combinatorial optimization problems where the goal is to find the best solution in a (possibly unknown) solution space. It uses the principle of biological evolution to generate successively better solutions from previous generations of solutions. The Memetic Algorithm (MA) is an extension of the GA which incorporates a local-search algorithm for each solution in between generations. According to Pastorino (2004), MA is able to improve convergence time, making it more favorable over GA.

5.3.2 Historical Background

MAs are a family of Metaheuristics that try to blend several concepts from tightly separated—i.e., their origins—families such as EAs and SA. The adjective ‘memetic’ comes from the term ‘meme’, coined by Dawkins (1976) to denote an entity that is analogous to the gene in the context of cultural evolution. The purpose of the analogy is to emphasize the departure from biologically inspired mechanisms of evolution, to more general processes where actual information is manipulated, learned, and transmitted. Due to the way in which this can be implemented, it is often the case that MAs are used under a different name (e.g., ‘hybrid EAs’, ‘Lamarckian EAs’,
etc.) and sometimes with a very restrictive meaning. At any rate, MA is a search strategy in which a population of optimizing agents synergically cooperates and competes (Moscato 1989). These agents are explicitly concerned with the knowledge from the problem being solved, as suggested by both theory and practice (Culberson 1998).

The characterization of a meme suggests that in culture evolution processes, information is not simply transmitting unaltered between individuals. In contrast, it is processed and enhanced by the communicating parts. This characteristic is accomplished in MAs by incorporating heuristics, approximate algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. Basically, most MAs can be regarded as a search strategy in which a population of optimizing agents cooperate and compete with each other (Moscato 1989).

MAs are described as a modified GA in which a local search plays a significant role (Moscato and Norman 1992). The local search function is added into a GA, thus applied to every offspring, which will be placed back into the population afterward. The MA can be perceived as a special kind of genetic search over the subspace of local optima. In general, the use of cross over and mutation without the local search will produce solutions outside the local optima space (Cheng and Gen 1996). With the local search in an MA, the solution is relocated into the optima space. This MA program coding comprises of two sections. The first one relates to the initialization of the population of parent schedules, while the second section is for the generation of the new schedule (offspring) from the parent schedule and the creation of the next generation of the population.
5.3.3 Working Principle

The 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 the 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, it is 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 the MA is to search and locate the local optimum more efficiently than the GA. Figure 5.2 explains the generic implementation of the Memetic Algorithm and Figure 5.3 explains the scheme of the Memetic Algorithm.

| Step 1: Encode Solution space. |
| Step 2: Set pop_size, max_gen, gen = 0; |
| Set cross_rate, mutate_rate; |
| Step 3: Initialize population |
| Step 4: While (gen < gensize) |
| Apply generic GA; |
| Apply local search; |
| End While; |
| Step 5: Apply final local search to best chromosome; |

Figure 5.2 The Memetic Algorithm
Figure 5.3 Scheme of the Memetic Algorithm
5.4 LOCAL SEARCH ALGORITHM (LS)

Local Search is a metaheuristic for solving computationally hard optimization problems. It can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) until a solution deemed optimal is found or a time bound is elapsed. It can be thought of as the process of an individual improving its idea of the solution.

It is adopted in an MA and is somewhat dependent on the problem being solved; however the common trait with any LS is that the parameters in the algorithm cannot be changed. This does not follow the definition of a meme, in that it can be changed because the adopted meme is the individual’s own interpretation of it. Furthermore, when memes are transmitted, changes to them are also passed on. Memes affect the behavior of an individual, and do not modify the genes themselves. However, as a practical issue, a meme in a Memetic Algorithm must be able to modify the genes in order to improve their fitness during the local search.

The termination of the local search can be on a time bound basis. Another common choice is to terminate it when the best solution found by the algorithm has not been improved in a given number of steps. The local search algorithms are typically incomplete algorithms as the search may stop even if the best solution found by the algorithm is not optimal. This can happen even if the termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithms.
5.4.1 Hill Climbing Local Search Algorithm

Hill Climbing is a mathematical optimization technique which belongs to the family of local search. It is relatively simple to implement, making it a popular first choice. The Hill Climbing local search algorithm is shown in Figure 5.4. It is a nature-based stochastic computational technique (Junying Chen et al 2005) and used as the local search to find better solutions in the neighborhood of the current solution produced by the GA in each iteration. When the termination condition is met, it returns with the best solution.

Hill Climbing is best suited to problems where the heuristic gradually improves, the closer it gets to the solution, and it works poorly where there are sharp drop-offs. It assumes that local improvement will lead to global improvement. It is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

| Step 1: Pick a random point in the search space. |
| Step 2: Consider all the neighbors of the current state. |
| Step 3: Choose the neighbor with the best quality and move to that state. |
| Step 4: Repeat Step 2 to 4 until all the neighboring states are of lower quality. |
| Step 5: Return the current state as the solution state. |

Figure 5.4 The Hill Climbing Local Search Algorithm

5.5 THE PROPOSED APPROACHES

Here, the GA is combined with the local search techniques like Hill Climbing, Tabu Search and Simulated Annealing for the DAG scheduling
problem explained in Chapter 3 which is implemented on a Cluster Of Workstations.

5.5.1 Encoding

To model the MA for the task scheduling problem, one of the issues is the genetic decoding which must be done before any program coding can be generated (Cheng and Gen 1996). The main concept of coding is to determine the representation of the machine setting and mapping it to the MA model.

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.

```
1     2     3
(2,1)  (3,2)  (1,1)
```

Figure 5.5 Chromosomal Representation
Each chromosome is represented as a group of genes i.e. the task-processor pair \((T_i, P_i)\) indicating that task \(T_i\) is assigned to the processor \(P_i\) shown in Figure 5.5. The position of genes in a chromosome represents the order in which the tasks should be executed. For example the following chromosomal representation shows 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 followed by task 1.

### 5.5.2 Initial Population

There are two parameters that have to be decided for initialization namely the initial population size and the procedure to initialize the population. There are two ways to generate the initial population namely the random initialization and the heuristic initialization. The initial chromosomes need not represent a legal solution.

#### Population size

The population size is one of the most important factors in the MA problems. The small population size may cause the solution to converge too quickly. If the population size is too large, computational time is wasted. Furthermore, while a large population size provides more chromosomes for testing, it also dilutes the fitness of the best chromosomes.

The population is created randomly i.e. a predefined number of chromosomes are generated, the collection of which forms the initial population. Most of the scheduling heuristics generate the initial population randomly with the necessary care on feasible solutions. Here the initial population is generated based on the priority calculation of the tasks at each level as shown in Table 5.1 (Gaussian Elimination Task Graph of 4 x 4 Matrix).
Fitness function

The fitness function will calculate the fitness value for each chromosome in the population and then store the value according to the individual chromosome. This value is also known as the objective value. 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.

5.5.3 Selection

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

5.5.4 Reproduction

Reproduction forms a new population of chromosomes by selecting chromosomes in the old population based on their fitness value through crossover and mutation. The crossover operator uses two chromosomes at a time to generate offspring by combining both chromosome features. The mutation operator uses only one chromosome altering one or more of its genes. In the conventional GA, crossover is used as the main genetic operator and mutation as a backup. The performance of the conventional GA depends strongly on how well the crossover function performs (Cheng and Gen 1996), while mutation helps to move the searching space.
Crossover

Crossover is the most important operator in the GA and MA since it implements the principle of evolution. It allows the MA to examine unvisited solution space and to create new chromosomes. The crossover operator selects any two chromosomes, namely parents, from the population and generates new chromosomes, namely offspring. New chromosomes are created with this operator by combining two selected parent chromosomes and swapping the 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

To escape from the local optimum, the mutation operation is created. 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 the locally best solution. The solution could stay in local optimum when many generations have passed and the chromosomes become more homogeneous and start to converge. The mutation operation prevents the solution from converging to local optimal solutions. The traditional mutation strategy based on job swapping is implemented in this study. Two positions are randomly selected and the alleles in these positions swap their values. To reduce the search space and computation time in the MA, the position swapping happens before the processor is assigned to the task. Therefore, there are no machines partitioning in the sequence in which the mutation strategy is the same as the single machine mutation strategy.
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. The probability of crossover and partial-gene mutation are not fixed in the proposed approaches.

**5.5.5 Local Search**

The Hill climbing search algorithm is a local search algorithm that iteratively performs a neighborhood search to pick the 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 Figure 5.6.

```
Step 1: Best solution ← initial solution;
Step 2: While (termination condition is not satisfied) do
    New solution ← neighbors (solution after cross over and mutation);
    If New solution is better than the best solution then
        Best solution ← New solution;
    End If;
End While;
```

**Figure 5.6 Modified Hill Climbing Local Search Algorithm**

Although more advanced algorithms may give better results, in some situations hill climbing works just as well. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to the optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.
5.5.6 Termination Criteria

There are three well-defined stopping criteria in the GA. The first stopping criterion is the maximum number of iterations. The second stopping criterion is the minimum objective value. The third stopping criterion is the value of the convergent constant on the objective value. This criterion is created to stop the unnecessary waste of the CPU time. This criterion is performed to terminate the MA when the solution cannot be changed for some number of consecutive iterations.

In this thesis, the second criterion is not implemented because of the normalize value technique. The objective value can reach the minimum point well before the MA reaches a good or optimal solution.

The proposed Memetic Algorithm is shown in Figure 5.7. Here the Local search can be Hill Climbing (or) Tabu Search (or) Simulated Annealing.

| Step 1: Generate population of size M based on task priority at each level using the attributes DTC, ACC, RPT, Rank and Priority Values for the given DAG. Calculate the fitness value of each Chromosome based on the earliest finish time of the last task using Equation (2.3). |
| Step 2: Select the fittest chromosome from the initial pool based on the Least Fitness Value (schedule length) using local search. |
| Step 3: Perform Crossover and Partial gene mutation with varying probabilities for the chromosomes obtained from Step 2. |
| Step 4: Evaluating the chromosomes obtained from Step 3 and forms a pool of the fittest Chromosomes using local search. |
| Step 5: Repeat Steps 3 and 4 until the termination criteria is met. |

Figure 5.7 The Proposed Memetic Algorithm
When no improvement solution has been found over the last \( n \) iterations, the algorithm terminates. Typically this value lies between 50 and 500 depending on the desired quality of the solution and the size of the problem.

The overall approach of Memetic Algorithm with Tabu Search (MA-TS) as local search enriches the searching behavior and avoids the premature convergence whose algorithm is presented in Figure 5.7. The DAG scheduling problem explained in Chapter 2 is implemented on a Cluster Of Workstations consisting of 35 HP Proliant machines.

5.6 RESULTS AND DISCUSSIONS

In this section, a number of experiments are carried out which outline the effectiveness of the proposed algorithm. The purpose of these experiments is to compare the performance of the Memetic Algorithm with the Genetic Algorithm for the task scheduling problem. Although the MA is a GA combined with the Hill Climbing (MA-LS), Simulated Annealing (MA-SA) and Tabu Search (MA-TS) 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 are tested on a cluster of workstations consisting of a 32 nodes HP Proliant cluster.

DAGs are generated randomly with different communication costs whose size varies from 10 to 50. A highly communication intensive application like the Gaussian elimination task graph is also generated with matrix size varying from 3 to 15. The results are compared for varying population sizes, where the size ranges from 5 to 200. Although the Memetic Algorithm is a GA combined with the Tabu Search as a local search, it is not necessarily the case that the genetic parameters are the most ideal for a Memetic Algorithm. The tasks are selected for an initial pool according to the
priority value as shown in Table 5.1 for the Gaussian Elimination task graph for a 4 x 4 matrix and are selected according to their fitness value.

Table 5.1 The Computed DTC, ACC, RPT, Rank and Priority Values for the Gaussian Elimination Task Graph 4 x 4 Matrix

<table>
<thead>
<tr>
<th>Task ID</th>
<th>ACC</th>
<th>RPT</th>
<th>DTC</th>
<th>Level</th>
<th>Rank</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>0</td>
<td>16</td>
<td>1</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
<td>5</td>
<td>2</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>19</td>
<td>6</td>
<td>2</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>19</td>
<td>7</td>
<td>2</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5.5</td>
<td>19</td>
<td>13</td>
<td>2</td>
<td>37</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3.5</td>
<td>28</td>
<td>27</td>
<td>3</td>
<td>58</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3.5</td>
<td>58</td>
<td>11</td>
<td>4</td>
<td>72</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4.75</td>
<td>58</td>
<td>12</td>
<td>4</td>
<td>74</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5.25</td>
<td>58</td>
<td>13</td>
<td>4</td>
<td>76</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3.75</td>
<td>72</td>
<td>25</td>
<td>5</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>4.5</td>
<td>100</td>
<td>10</td>
<td>6</td>
<td>114</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>4.25</td>
<td>114</td>
<td>12</td>
<td>6</td>
<td>130</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>114</td>
<td>29</td>
<td>7</td>
<td>147</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>4.75</td>
<td>147</td>
<td>10</td>
<td>8</td>
<td>161</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>5.25</td>
<td>161</td>
<td>0</td>
<td>9</td>
<td>166</td>
<td>1</td>
</tr>
</tbody>
</table>

In the proposed approach, the effects of the varying population size and the number of iterations are investigated and the results are depicted in Figures 5.8 and 5.9. The graphs show the performance of the proposed Memetic Algorithms for the Gaussian Elimination task graph of size 4×4 matrix. It is evident that the performance of the proposed approach (MA-TS) improves with the population size increased.
The number of iterations is increased for a given population size. At a minimum level of iteration itself the optimum fitness value is reached. As the iterations are increased by applying more swaps, many possible chromosomes are produced. Thereby the fitness value would be very much reduced. After a particular iteration value, the graph shows the similar performance for the proposed Memetic Algorithm approaches shown in Figure 5.9.

The performance of the MA-TS is almost the same as that of the MA-LS approach irrespective of the size of the Gaussian Elimination Task Graph. Figure 5.9 shows the performance of the low communication graph (CCR=0.1); the behavior of the MA-TS is very low compared to the MA-LS. The reason for the comparatively lesser scheduled length is the fact that TS has identified a smaller working time processor sequence during its method of swap and relocate. The characteristic of the TS is that it climbs rapidly to the nearest optima and then is subject to the tabu list constraints searches for other optima in the region. Intuitively, the large neighborhood space directs the TS to search more of the neighborhood of the first local optima.
The chief limitation of a hill climbing procedure is that the local optimum obtained at its stopping point, when no improving moves are possible, may not be a global optimum. TS guides such a heuristic to continue exploration without becoming confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged. The performance of the proposed Memetic Algorithms for $5 \times 5$ Gaussian Elimination task graph is shown in Table 5.2.

**Table 5.2 Performance of the proposed Memetic Algorithms**

<table>
<thead>
<tr>
<th>No.of Iterations</th>
<th>Population = 10</th>
<th>Population = 20</th>
<th>Population = 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>330</td>
<td>326</td>
<td>261</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>189</td>
<td>205</td>
</tr>
<tr>
<td>15</td>
<td>155</td>
<td>155</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>145</td>
<td>139</td>
<td>192</td>
</tr>
<tr>
<td>50</td>
<td>139</td>
<td>136</td>
<td>170</td>
</tr>
<tr>
<td>100</td>
<td>139</td>
<td>137</td>
<td>157</td>
</tr>
<tr>
<td>120</td>
<td>137</td>
<td>137</td>
<td>147</td>
</tr>
</tbody>
</table>
MA converges very fast when compared to GA as shown in Figure 5.10. The results of both MA and GA are compared by varying the number of iterations from 5 to 250 for the Gaussian Elimination 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 searches and uses GA to perform exploration while the local search performs the exploitation.

Figure 5.11 Comparison with Existing Heuristics
The proposed Memetic Algorithms (MA-LS, MA-SA and MA-TS) are compared with the classical list scheduling algorithms like EFT, CPOP and PETS algorithm for the Gaussian Elimination task graph whose matrix size is $5 \times 5$. The performance of the MA is compared with the existing EFT, CPOP and PETS algorithm and is shown in Figure 5.11. From the above results, it is clear that the proposed MA performs well when compared to the other list scheduling algorithms. Since the three algorithms are based on list scheduling, the method producing the scheduling list and the priority assigning rules are different.

The EFT 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 the CC, DTC and RPT values [4, 5]. But in the proposed MA, first the chromosomes are encoded as task-processor pair or as a schedule solution. Task priorities are calculated as in the 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 the local search. The fitness value of that best chromosome gives the schedule length.

For low communication graph (CCR=0.1), the results are the same initially but at the particular value of iteration, the behavior of the MA-TS is very low compared to MA-LS. The improvement in makespan produced by MA-TS is 1.3% when compared to MA-LS, 3.85% when compared to SLTS (proposed in Section 2), 6.25% when compared to CPOP and 11% when compared to EFT.
5.7 SUMMARY OF THE CHAPTER

In this chapter, by hybridizing the population based evolutionary searching ability of GA with local improvement abilities of Hill Climbing (HC) and Tabu Search (TS) to balance exploration and exploitation, effective approaches have been proposed for Task Scheduling with the objective of minimizing the makespan. It is appropriate for scheduling highly communication intensive applications on a cluster of workstations considering communication contention.

The proposed approach MA-TS is compared with the Memetic Algorithm using Hill Climbing, Simulated Annealing as the local search as well as with the GA. The experimental results show that the proposed approach gives a better solution compared to the other two because the TS avoids premature convergence compared to the HC as the local search. It is important to device a scheduling algorithm flexible enough to react to the dynamic change of the CPU Load and the data locality of the tasks. To analyze them dynamically in the nodes of the cluster and to schedule the tasks accordingly, a Resource Aware Scheduler is developed.