CHAPTER 4

PROPOSED HEURISTIC BASED HYBRID METAHEURISTICS

4.1 INTRODUCTION

The multiprocessor task scheduling problem belongs to the class of NP hard problem as solution of even small size problems is more difficult and complex. The multiprocessor task scheduling with optimization of more than one objective under the constraint of communication delay also becomes NP hard with greater complexity towards optimality in a reasonable time. Several exact, heuristics and randomized methods have been developed by different researchers for the solution of NP hard problems. It has been accepted that finding optimality to NP hard problems is not a viable option since large amount of computational time is needed for judgment of optimal solutions. In reality, a good initial solution can be obtained by a heuristic in a reasonable computational time.

![Diagram of Scheduling Heuristics]

Figure 4.1: Classifications of Scheduling Heuristics

Figure 4.1 shows the broad classification of heuristics such as dispatching rules, constructive and improvement heuristics. Dispatching rules are the most classic and well known methods for building the schedule. These rules are very often used in practice and are mostly used for obtaining initial sequence in some improvement heuristics.
Constructive heuristics build a schedule from the scratch by making a series of passes through the list of unscheduled jobs where at each pass one or more jobs are selected and added to the schedule. Contrary to constructive heuristics, improvement heuristics start from an existing solution and apply some improvement procedure.

The constructive heuristics are further classified into three categories, i.e. list scheduling, clustering based and duplication based heuristics.

**List Scheduling Heuristics:** List scheduling based heuristics are the most primeval that arrange the nodes of the graphs in the form of a list which is based on some priority. The task having maximum priority is selected and schedule on the most suitable processor. The list scheduling can be static or dynamic based on the priority calculation phase (compile time or run time). A huge number of the list scheduling heuristics have been developed and available for task scheduling problem on homogeneous and heterogeneous multiprocessor architecture. Some of the list scheduling algorithms are: ISH (Kruatrachue and Lewis; 1987), HLFET (Adam et. al.; 1974), MCP (Wu and Gajski; 1990), ETF (Hwang et al.; 1989), DLS (Sih and Lee; 1993)

**Clustering based heuristics:** Clustering based scheduling is also the most widely studied and exploited heuristics for the multiprocessor task scheduling. A cluster consists of a two phase heuristic, in the first phase, tasks that must schedule together on the same processor, owing to their higher inter processor communication costs are decided upon and mapped to a common cluster under the ambience of cluster heuristics. There by generating an unbounded number of clusters for the whole task graph. In the second phase, these clusters are mapped onto the physical processors and the tasks with in a cluster are sequentialized using a priority based approach. Some of these algorithms are: LC (Kim and browne; 1988), EZ (Sarkar; 1989) etc.

**Duplication based heuristics:** The duplication based heuristics are more efficient for fine grain tasks graphs and networks with increased communication latencies. Most of the duplication algorithms are designed under the assumption of the unbounded availability of fully connected processors and lies in the increased complexity range. Low complexity optimal duplication algorithms work under restricted cost or shape parameters for the task graphs (Bansal et al.; 2003). Some of these algorithms are: DSH (Kruatrachue and Lewis; 1987), BTDH (Chung and Ranka; 1992), CPFD (Ahmad and Kwok; 1998) etc.

Therefore, the list scheduling algorithms are simple and most commonly used algorithms for multiprocessor task scheduling problem as being well suited for each category of task
In the present work, five well known list heuristics i.e. HLFET (Highest Level First Estimated Time), ISH (Insertion Scheduling heuristic), MCP (Modified critical Path), ETF (Earliest Time First), DLS (Dynamic level Scheduling) have been considered for multiprocessor task scheduling on homogeneous processors.

4.2 LIST SCHEDULING HEURISTICS

In List scheduling heuristics, each task is allotted a priority and then appended to a queue of waiting tasks in order of their increasing/decreasing priority. As processors become available, a task with the highest priority is selected and allotted to the most suitable processor. General outline of list scheduling is:

Step 1 Compute priority for all tasks by traversing the application DAG.;
Step 2 Sort the tasks in a list by decreasing/increasing order of priority;
Step 3 While there are unscheduled tasks on the list do
  Step 4 Remove the tasks from the list one by one;
  Step 5 For each processor do
    Step 6 Compute the earliest start time;
    Step 8 Find the minimum earliest start time processor P;
  Step 9 Assign task to processor P;
Step 10 End while

The several algorithms based on list scheduling differ in the way of the priorities of tasks that are being determined. There are several methods for determining the priorities of tasks, such as HLF (Highest Level First), LP (Longest Path), LPT (Longest Processing Time) and CP (Critical Path) etc.

Two frequently used attributes for assigning priorities are the t-level (top level) and b-level (bottom level). The t-level of a node $n_i$ is the length of a longest path (there can be more than one longest path) from an entry node to $n_i$ (excluding $n_i$). Here, the length of a path is the sum of all the nodes and edge weights along the path. The b-level of a node $n_i$ is the length of a longest path from $n_i$ to an exit node. A critical path (CP) of a DAG, which is an important structure in the DAG, is a longest path in the DAG. Some scheduling algorithms do not take into account the edge weights in computing the b-level and call it the static b-level or simply static level (sl). Another attribute used in scheduling algorithms for assigning priority is ALAP (As late as possible). The ALAP start time of a
node is a measure of how far the node’s start time can be delayed without increasing the schedule length. Algorithms (Kwok and Ahmad; 1999) for computing t-level, b-level and ALAP are as follows:

(a) **Algorithm to compute t-level**

**Step 1** Construct a list of nodes in topological order in the list.

**Step 2** for each node \( n_i \) in List do

**Step 3** \( \text{max} = 0 \)

**Step 4** for each parent \( n_x \) of \( n_i \) do

**Step 5** if \( t\text{-level}(n_x) + w(n_x) + c(n_x, n_i) > \text{max} \) then

**Step 6** \( \text{max} = t\text{-level}(n_x) + w(n_x) + c(n_x, n_i) \)

**Step 7** endif

**Step 8** endfor

**Step 9** \( t\text{-level}(n_i) = \text{max} \)

**Step 10** endfor

(b) **Algorithm to compute b-level**

**Step 1** Construct a list of nodes in reverse topological order in RevList.

**Step 2** for each node \( n_i \) in RevList do

**Step 3** \( \text{max} = 0 \)

**Step 4** for each child \( n_y \) of \( n_i \) do

**Step 5** if \( c(n_i, n_y) + b\text{-level}(n_y) > \text{max} \) then

**Step 6** \( \text{max} = c(n_i, n_y) + b\text{-level}(n_y) \)

**Step 7** endif

**Step 8** endfor

**Step 9** \( b\text{-level}(n_i) = w(n_i) + \text{max} \)

**Step 10** endfor

(c) **Algorithm to compute ALAP**

**Step 1** Construct a list of nodes in reverse topological order in RevList.

**Step 2** for each node \( n_i \) in RevList do

**Step 3** \( \text{min} = \text{CP}_\text{Length} \)

**Step 4** for each child \( n_y \) of \( n_i \) do

**Step 5** if \( \text{alap}(n_y) - c(n_i, n_y) < \text{min} \) then
Step 6 \( \text{min} = \text{alap}(n_y) - c(n_i, n_y) \)

Step 7 \( \text{endif} \)

Step 8 \( \text{endfor} \)

Step 9 \( \text{alap}(n_i) = \text{min} - w(n_i) \)

Step 10 \( \text{endfor} \)

4.2.1 Highest Level First with Estimated Time (HLFET) Algorithm

The Highest Level First with Estimated Time (HLFET) algorithm assigns scheduling priority to each node based on static b-level, which is the length of a longest path from the node to an exit node without considering the length of the edge (Jin et al.; 2008). HLFET is one of the simplest List Scheduling algorithm and is outlined below:

Step 1 Calculate the static b-level (i.e., sl or static level) of each node.

Step 2 Make a ready list in a descending order of static b-level. Initially, the ready list contains only the entry nodes. Ties are broken randomly.

Repeat

Step 3 Using the non insertion approach; schedule the first node in the ready list to a processor that allows the earliest execution.

Step 4 Update the ready list by inserting the nodes that are now ready.

Until all nodes are scheduled.

As HLFET uses non insertion approach, i.e. an idle time slot is not utilized, which affects the performance of the algorithm.

4.2.2 Insertion Scheduling Heuristic (ISH) Algorithm

The Insertion Scheduling Heuristic (ISH) algorithm improves the HLFET algorithm by utilizing the idle time slots in the schedule. Initially, ISH uses the similar procedure as HLFET for creating a ready list based on the static b-level and schedule the first node in the ready list using the non insertion approach. Once the scheduling of the node creates an idle slot, ISH checks if any task in the ready list can be inserted into the idle slot but cannot be scheduled earlier on the other processors. Schedule such tasks as much as possible into the idle slot (Jin et al.; 2008). A basic outline of the algorithm is given below:

Step 1 Compute the static b-level of each node.
Step 2 Make a ready list in a descending order of static b-level. Initially, the ready list contains only the entry nodes. Ties are broken randomly.

Repeat

Step 3 Using the non insertion algorithm; schedule the first node in the ready list to the processor that allows the earliest execution.

Step 4 If scheduling of this node causes an idle time slot, then find as many nodes as possible from the ready list that can be scheduled to the idle time slot but cannot be scheduled previously on other processor.

Step 5 Update the ready list by inserting the nodes that are now ready.

Until all nodes are scheduled.

4.2.3 Modified Critical Path (MCP) Algorithm

The MCP algorithm computes the ALAPs of all the nodes and constructs a list of nodes in an ascending order of ALAP times. Ties are broken by considering the ALAP times of the children of a node. The MCP algorithm then schedules the nodes on the list one by one such that a node is scheduled to a processor that allows the earliest start time using the insertion approach. The algorithm is described as:

Step 1 Compute the ALAP time of each node.

Step 2 For each node, create a list which consists of the ALAP times of the node itself and all its children in a descending order.

Step 3 Sort these lists in an ascending lexicographical order. Create a node list according to this order.

Repeat

Step 4 Using the insertion approach; schedule the first node in the node list to a processor that allows the earliest execution.

Step 5 Remove the node from the node list.

Until the node list is empty.

4.2.4 Earliest Time First (ETF) Algorithm

The ETF algorithm computes the earliest start times for all ready nodes at each step and selects the one with the smallest start time. The algorithm is:

Step 1 Compute the static b-level of each node.

Step 2 Initially, the pool of ready nodes includes only the entry nodes.
Repeat

Step 3 Calculate the earliest start-time on each processor for each node in the ready pool. Pick the node processor pair that provides the earliest time using the non insertion approach. Ties are broken by selecting the node with a higher static b-level. Schedule the node to the corresponding processor.

Step 4 Add the newly ready nodes to the ready node pool.

Until all nodes are scheduled.

4.2.5 Dynamic Level Scheduling (DLS) Algorithm

The DLS algorithm make use of an attribute called dynamic level (DL), which is the difference between the static b-level of a node and its earliest start time on a processor. At each scheduling step, the algorithm computes the DL for every node in the ready pool on all processors. The node processor pair which provides the higher value of DL is selected for the scheduling. The description of the algorithm is given below:

Step 1 Calculate the b-level of each node.

Step 2 Initially, the ready node pool includes the entry nodes only.

Repeat

Step 3 Determine the earliest start time for every ready node on each processor. Compute the DL for every node processor pair by subtracting the earliest start time from the node’s static b-level.

Step 4 Select the node processor pair that gives the larger DL. Schedule the node to the corresponding processor.

Step 5 Add the newly ready nodes to the ready pool.

Until all nodes are scheduled.

4.3 METAHEURISTICS

Meta-heuristics are the approaches that perform a typically incomplete search in the solution space by iteratively creating and evaluating new candidate solutions. The most prominent and successful metaheuristics are described as:

**Standard local search:** Standard local search tries to improve on it by iteratively deriving a similar solution in the neighborhood of the so-far best solution and initiate from the solution created at random or by some problem specific heuristic. Move operator
is used for the judgment of neighboring solution and must be carefully defined as per the nature of the problem. The major disadvantage of standard local search is its high probability of getting trapped at a poor local optimum. It applies standard local search multiple times from different starting solutions and returns the best local optimum identified.

**Simulated annealing:** It is another way of enabling the local search for preventing from the local optima. Sometimes simulated annealing also accepts inferior neighboring solutions as uses the probabilistic way of selecting the neighboring solutions. At the beginning of the optimization, worse solutions are accepted with a relatively high probability which is reduced over time in order to achieve convergence.

**Tabu search:** This strategy extends local search by the introduction of memory. Stagnation at a local optimum is avoided by maintaining a data structure called as history, in which the last created solutions or alternatively the last moves (i.e., changes from one candidate solution to the next) are accumulated. The solutions, respectively moves, are forbidden (tabu) in the next iteration and the algorithm is forced to approach unexplored areas of the search space.

**Evolutionary algorithms (EAs):** They are the broader class of meta-heuristics based on the common idea of adopting principles from natural evolution in the simplified ways. The genetic algorithm is the common example of evolutionary algorithm which tries to mimic the natural evolution process and generally start with an initial population of individuals, which can either be generated randomly or based on some heuristic. Among the metaheuristics, Genetic Algorithm (GA) and Simulated Annealing (GA) have been considered for multiprocessor task scheduling.

### 4.3.1 Genetic Algorithm

Genetic algorithm introduced by John Holland (1975) is an optimization technique based on the natural evolution, which includes the survival of the fittest idea into a search algorithm and provides a method of searching, which does not need to explore every possible solution in the feasible region for obtaining the best result. In nature, the fittest individuals are most likely to survive and mate; therefore the next generation should be fitter and healthier because they were bred from healthy parents. The same idea applies to the problems by first ‘guessing’ solutions and then combining the fittest solutions to create a new generation of solutions, which should be better than the previous generation.
4.3.1.1 Features of the Genetic Algorithm

- Genetic algorithms work well for extensively large optimization problems.
- Genetic algorithms work as solution providers for various problems.
- It cracks problems with multiple in-hand solutions.
- Genetic algorithm is a method which is very easy to understand and to implement.
- It practically does not demand the expertise of mathematics.
- Genetic algorithms are easily transferred to existing simulations and models.
- Handles noisy functions well.
- GAs are the computational model of innovation and creativity.
- GA follows random modeling instead of using deterministic rules (Gupta; 2013)

4.3.1.2 Genetic Algorithm Terminology

Fitness Function
The fitness function is the function that has to be optimized. This is known as the objective function for the standard optimization algorithms.

Individuals
An individual or chromosome is any point by which the fitness function is applied. The value of the fitness function for an individual is known as score.

An individual is sometimes referred to as a genome and the vector entries of an individual as genes. A locus is the position of a gene and Alleles are the values of the gene.

Population and Generations
A population is an array of individuals. For example, if the size of the population is 20 and the number of variables in the fitness function is 4, represent the population by a $20 \times 4$ matrix. The same individual can appear more than once in the population. For example, the individual $(1, 3, 2, 4)$ can appear in more than one row of the array. At every iteration, the genetic algorithm performs a series of computations on the current population for generating the new population. Each successive population is called a new generation.

Fitness Values and Best Fitness Values
The fitness value of an individual is the value of the fitness function for the particular individual. The best fitness value of a population is the smallest fitness value for any individual in the population.
Parents and Children
To create the next generation, the genetic algorithm selects certain individuals in the current population called parents and uses them for creating the individuals in the next generation called children. Generally, the algorithm is more likely to select parents that have better fitness values.

4.3.1.3 Outline of Genetic algorithm

Step 1 Create a random initial population.

Repeat

Step 2 Score each member of the current population by computing its fitness value.

Step 3 Scale the raw fitness scores for converting them into a more practical range of values.

Step 4 Selects members called parents based on the fitness.

Step 5 Some of the individuals in the current population that have lower fitness are chosen as an elite. These elite individuals are passed to the next population.

Step 6 Create children from the parents. Children are produced either by making random changes to a single parent—mutation—or by combining the vector entries of a pair of parents—crossover.

Step 7 Replace the current population with the children to form the next generation.

Until the stopping criteria is met.

4.3.1.4 Parameters of the Genetic Algorithm

- Population Size (Ps): Population size refers to the search space, i.e. algorithm has to search the specified number of sequences and larger the sequences, more the time is needed to execute the process of genetic algorithm and with smaller value algorithm would not be able to explore the search space. So, a mechanism is needed to set the value of population size.
• Selection function: Selection function specifies how the genetic algorithm chooses parents for the next generation. Commonly used selection functions are uniform, tournament & roulette wheel.

• Crossover function: Crossover is the breeding of two parents to produce a single child. That child has features from both parents and thus may be better or worse than either parent as per fitness function. Analogous to natural selection, the more fit the parent, the more likely the generation has. Mainly there are two types of crossover: one point crossover & two point crossover. Commonly used two point crossovers are order crossover (OX), partially matched crossover (PMX), Cycle crossover (CX) and position based crossover. Different types of crossover can be used depending upon the type and nature of the problem.

• Mutation function: For each sequence in the parent population a random number is picked and by giving this sequence a percent chance of being mutated. If this sequence is picked for mutation, then a copy of the sequence is made. Commonly used mutation functions are swap, insert, scramble, inversion. Different types of mutation can be used depending upon the type & nature of the problem.

• Elite Count: The best sequences found should be considered in subsequent generations. At a low, the only best solution of the parent generation needs to be imitated to the next generation, thus ensuring the best score of the next generation is at least as good as the previous generation. Here elite is expressed as a number of sequences. Value of elite count also affects the results of the GA.

• Crossover fraction: It is the fraction for which crossover has to perform on the parents as per population size in each generation. This is also an important parameter which affects the performance of GA & a mechanism is needed to set its value.

• Stopping condition: Stopping condition determines what causes the algorithm to terminate. There are a lot of options which are described below

  Generations: Specifies the maximum number of iterations for the genetic algorithm to perform.
  Time limit: Specifies the maximum time in seconds the genetic algorithm runs before stopping.
  Fitness limit: The algorithm stops if the best fitness value is less than or equal to the value of Fitness limit.
Stall generations (StallGenLimit): The algorithm stops if the weighted average change in the fitness function value over Stall generations is less than Function tolerance.

Stall time limit (StallTimeLimit): The algorithm stops if there is no improvement in the best fitness value for an interval of time in seconds specified by Stall time.

Function tolerance (TolFun): The algorithm runs until the cumulative change in the fitness function value over Stall generations is less than or equal to Function Tolerance.

4.3.2 Simulated Annealing

Simulated annealing (SA), one of the meta-heuristic algorithm, was introduced for solving combinatorial optimization problems by Kirkpatrick et al. (1983). It is a neighborhood search procedure that produced good results for NP hard problems. A classical simulated annealing begins by generating an initial solution randomly. At each stage, the new solution taken from the neighborhood of the current solution is accepted if it has a lower or equal cost; if it has a higher cost it is accepted with a probability that decreases as the difference in the cost increases and as the temperature of the method decreases. This temperature, which is simply a positive number, is periodically reduced by some temperature scheme, so that it moves gradually from a relatively high value to lower value as the method progresses. Thus, at the start of SA, most worsening moves are accepted, but at the end only improving ones are likely to be accepted. The method converges to a local optimum as the temperature approaches zero, because SA has performed many perturbations at a high temperature, which have pushed the search path into new areas, a better local optimum solution should hopefully be reached.

4.3.2.1 Features of Simulated Annealing

Some of the features of the simulated annealing (Busetti; 2003) are:

- Simulated Annealing can deal with arbitrary systems and cost functions.
- It statistically guarantees finding an optimal solution.
- It is relatively easy to code, even for complex problems.
- It generally gives a "good" solution.
- It is a robust and general technique.
- Its main advantages over other local search methods are its flexibility and its ability to approach global optimality.
The algorithm is quite versatile since it does not rely on any restrictive properties of the model.

4.3.2.2 Outline of Simulated Annealing Algorithm

Step 1: Generate a random initial solution and initialize the initial temperature.

Repeat

Step 2: The annealing function will modify this schedule and return a new schedule that has been changed by an amount proportional to the temperature.

Step 3: The algorithm determines whether the new schedule is better or worse than the current based on their objective function value. If new is better than the current, it becomes the next schedule otherwise the algorithm makes it the next schedule based on an acceptance probability.

Step 4: The algorithm lowers the temperature proportional to cooling rate and store the best point found so far.

Step 5: Re-annealing is performed after a certain number of schedules are accepted by SA. Re-annealing raises the temperature and the search is resumed with the new temperature values.

Until stopping criteria is met.

4.3.2.3 Parameters of Simulated Annealing

- **Initial Temperature**: The temperature is the control parameter in simulated annealing that is decreased steadily as the algorithm proceeds. It determines the probability of accepting an inferior solution at any step and used for bounding the extent of the search in a given dimension.

- **Annealing Function**: The annealing function modifies current schedule and return a new schedule that has been changed by an amount proportional to the temperature. Different types of annealing function can be used depending upon the type and nature of the problem.

- **Temperature Function**: It updates the temperature vector for annealing process. Different types of temperature functions can be used depending upon the type & nature of the problem.

- **Acceptance Function**: It is used to determine whether to accept a change that leads to a worsening of the objective function value. SA randomizes the search procedure to allow for occasional changes that worsen the solution, in an attempt
to reduce the probability of the search becoming trapped in a locally optimal solution. Various types of acceptance functions can be used depending upon the type & nature of the problem.

- **Re-anneal Interval**: Annealing is the technique of closely controlling the temperature when cooling a material to ensure that it is brought to an optimal state. Re-annealing raises the temperature after a certain number of new points have been accepted and starts the search again at the higher temperature. Re-annealing avoids getting caught in local minima.

- **Stopping condition**: Stopping condition is used to stop the algorithm after certain numbers of runs. The following conditions can be used to determine when to stop:
  
  - TolFun — The algorithm runs until the average change in value of the objective function in StallIterLim iterations is less than the value of TolFun.
  - MaxIter — The algorithm stops when the number of iterations exceeds this maximum number of iterations.
  - MaxFunEval specifies the maximum number of evaluations of the objective function. The algorithm stops if the number of function evaluations exceeds the value of MaxFunEval.
  - TimeLimit specifies the maximum time in seconds the algorithm runs before stopping.
  - ObjectiveLimit — The algorithm stops when the best objective function value is less than or equal to the value of ObjectiveLimit.

### 4.4 PROPOSED HEURISTIC BASED HYBRID METAHEURISTICS

Hybrid algorithms for scheduling are inspired by the fact that each type of scheduling technique has its own strengths and weaknesses while appearing complementary to each other (Wen et al.; 2011) Heuristic based algorithms are almost as efficient, but do not produce consistent results on a wide range of problems, especially when the complexity of the scheduling problem increases. Metaheuristic based scheduling algorithms obtain schedules of better quality at the expense of more computing efforts due to limited exploration ability. Therefore, combining the advantages of both heuristic and metaheuristic algorithms is the interesting choice of researchers for better results. So, in the present work five list scheduling heuristics i.e HLFET, ISH, MCP, ETF and DLS are
hybridized with genetic algorithm and simulated annealing for bi-criteria multiprocessor task scheduling problem with precedence constraints on homogeneous processors.

4.4.1 Hybrid Genetic Algorithm (HGA)

In Genetic algorithms, methods for creating an initial population are varied: feasible only, randomized, using heuristics, etc. Classical GA generates initial population randomly and drawbacks of the algorithm is that, the choice of the initialization procedure has an important influence on the quality of solution and a better initial solution might provide better results. Due to the large search space in multiprocessor task scheduling, it is expected that random generation of initial solutions provides relatively weak results. For this, initial solution is obtained by the application of list scheduling heuristics for finding near to optimal results in a very reasonable time. The detailed description of the proposed hybrid genetic algorithm is given below:

Step 1 Encoding
a) Encoding gives the representation of a chromosome. In the present work, chromosome is represented as \((T, P)\) pair where \(T\) is task sequence \(t_1, t_2, \ldots, t_n\) & \(P\) is allocated processor sequence \(p_1, p_2, \ldots, p_n\).

b) Each task sequence is a permutation of task numbers & each processor sequence is a permutation of processor numbers \((1, 2, \ldots, m)\) with length equal to a number of tasks.

Step 2 Initialization

Each task sequence is a permutation of task numbers, so each task will be processed according to its appearance. As dependency exists between tasks, so each task in the task sequence should appear before all of its children and after all of its parents. Therefore, some permutations of the tasks may not be valid and some mechanism would be needed to validate the invalid sequences. The initial population in the present work is generated using the following steps:

1. Use scheduling heuristic (table 1) for generating the seed sequence.
2. Generate \((p_s - 1)\) population randomly using the following steps
   a) Generate the valid task sequences (TS) of \((p_s-1)\) using the algorithm as stated by Bonyadi and Moghaddam (2009).
   b) Generate the processor sequences (PS) of \(p_s-1\) randomly.
c) Map the each task sequence (T) from TS to randomly selected processor sequence (P) from PS giving each chromosome in the form (T, P) i.e task sequence followed by mapped processor sequence.

3. Combine the seed sequence with randomly generated population as per population size (ps).

Step 3 Reproduction

Different reproduction operators are used for task and processor sequences due to different nature. The sequences are firstly separated from a chromosome and then used individually for performing crossover and mutation. Generated task sequences after reproduction may not be valid in terms of dependency, so a mechanism is used for validating the task sequences as stated by Bonyadi and Moghaddam (2009). Then valid task sequences after reproduction (TS′) are mapped to processor sequences after reproduction (Ps′) based on minimum fitness value.

The algorithm uses the following steps for generating the new offsprings:

a) Scores each member of the current population by computing fitness (i.e. weighted sum of makespan and total completion time).

b) Best Individuals as per fitness in the current population act as elite and concede in the next population.

c) Selects parents based on the fitness value as per selection function for reproduction.

d) Replace the current population with children to form next generation.

Step 4: Stopping criteria

The algorithm stops when one of stopping criteria is met.

Generation of seed sequence as in step 1 of HGA has been obtained from the one of the five list scheduling heuristics as shown in table 4.1. Therefore five heuristic based HGAs called as HLFET-GA, ISH-GA, MCP-GA, ETF-GA, DLS-GA, have been proposed and developed for comparative analysis amongst them and other methods.
Table 4.1: Different Heuristics for HGAs and HSAs.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>HLFET</td>
<td>HLFET decides the priorities of tasks according to static level and allocates processor to the highest priority task based on minimum start time.</td>
</tr>
<tr>
<td></td>
<td>(Adam et al.; 1974)</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>ISH</td>
<td>ISH is similar to HLFET except it attempts to fill the idle time slots created in the partial schedule by scheduling other nodes into them.</td>
</tr>
<tr>
<td></td>
<td>(Kruatrachue and Lewis; 1987)</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>MCP</td>
<td>The MCP algorithm uses the ALAP of a node as scheduling priority and uses the ALAP of child nodes to break the ties. It allocates the processor to highest priority process based on minimum start time using insertion approach.</td>
</tr>
<tr>
<td></td>
<td>(Wu and Gajski; 1990)</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>ETF</td>
<td>ETF algorithm computes the earliest start time of all the ready nodes by examining the start time of node on all processors exhaustively and selects the one with smallest start time.</td>
</tr>
<tr>
<td></td>
<td>(Hwang et al.; 1989)</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>DLS</td>
<td>At each scheduling step, the algorithm computes the DL (difference between static level of a node and its earliest start time) for every node in the ready pool on all processors. The node-processor pair which gives the largest value of DL is selected for the scheduling.</td>
</tr>
<tr>
<td></td>
<td>(Sih and Lee; 1993)</td>
<td></td>
</tr>
</tbody>
</table>

4.4.2 Hybrid Simulated Annealing (HSA)

In SA, methods of creating the sample schedule also vary. In the present work one of the five list scheduling heuristic (table 4.1) is used to give the sample schedule in classical simulated annealing algorithm resulting into five HSAs i.e HLFET-SA, ISH-SA, MCP-SA, ETF-SA, DLS-SA for comparison among themselves. Basic outline of hybrid simulated annealing is given below:

**Step 1: Encoding**

c) Encoding give the representation of a schedule. In the present work, a schedule is represented as (T, P) pair where T is task sequence t₁, t₂,......,tₙ & P is allocated processor sequence p₁,p₂,.....,pₙ.

d) Each task sequence is a permutation of task numbers & each processor sequence is a permutation of processor numbers (1, 2... m) with length equal to a number of tasks.

**Step 2: Initialization**

The algorithm begins with an initial seed sequence obtained by one of five list scheduling heuristics as shown in table 4.1.

**Step 3: Neighborhood generation (move function) and acceptance:**

It is a method of generating new schedules and different heuristics for neighborhood generation vary significantly. The basic mechanism of neighborhood generation contains the following steps:
a) Separate the task sequence (T) from a processor sequence (P) in a schedule. Task sequences follow dependency constraint, but no such constraint is required for processor sequence. So, both sequences should be treated individually.
b) Apply the suitable neighborhood generation method for both sequences individually for providing new task sequence (T’) & processor sequence (P’).
c) As new task sequence (T’) may not be valid in terms of dependency, so a mechanism is used for validating the task sequence (T’) using the algorithm of validation as stated by Bonyadi and Moghaddam (2009) resulting into validated task sequence (T’’).
d) The validated task sequence (T’’) is combined to processor sequence (P’).

Now if the new schedule is better than the current in terms of the objective function, then it becomes the next schedule, otherwise the probability of acceptance is less than 50%.

**Step 4:** Cooling strategy (Temperature Function): It is related to systematically lowering the temperature, which is proportional to the cooling rate and store the best point found so far.

**Step 5:** Re-annealing: Re-anneal Interval is a parameter of simulated annealing which raises the temperature after the algorithm accepts a certain number of new points and starts the search again at the higher temperature.

**Step 6:** Stopping limit: The algorithm stops when one of stopping criteria is met.

### 4.4.3 Hybrid Genetic Algorithm and Simulated Annealing (GASA)

Due to the large search space in multiprocessor task scheduling, it is expected that random generation of initial solutions provides relatively weak results. For this, initial solution is obtained by application of GA for finding near to optimal results in a very reasonable time and final optimal solution by SA procedure and called as hybrid GASA.

Outline of GASA described as follows:

1. Generate ps (population size) population randomly.

2. The algorithm then creates a sequence of new populations. At each step, the algorithm uses the individuals in the current generation to create the next population. To create the new population, the algorithm performs the following steps:
   a) Scores each member of the current population by computing fitness.
   b) Selects members, called parents, based on their fitness.
c) Some of the individuals in the current population that have lesser fitness are chosen as an elite. These elite individuals are conceded to the next population.

d) Produces offspring from the parents. Offsprings are produced either by combining the vector entries of a pair of parents—crossover or by making random changes to a single parent—mutation.

e) Replaces the current population with the children to form the next generation.

3. The algorithm generates a sequence called as \( S \) when the maximum number of generations reaches 50 which is the seed sequence in SA procedure.

4. Obtain the fitness function for \( S \)

5. Initialize the initial temperature and Reanneal interval.

6. The annealing function will then modify this schedule and return a new schedule that has been changed by an amount proportional to the temperature.

7. The algorithm determines whether the new schedule \( S' \) is better or worse than the current \( S \). If \( S' \) is better than the \( S \), it becomes the next schedule \( S=S' \). If the \( S' \) is worse than the \( S \), the algorithm may still make it the next schedule based on an acceptance probability.

8. The algorithm lowers the temperature proportional to cooling rate and store the best point found so far.

9. Re-annealing is performed after a certain number of schedules are accepted. Re-annealing raises the temperature and the search is resumed with the new temperature values.

10. The algorithm stops when the maximum number of iterations reaches 50.

The next chapter presents the parameter optimization of different metaheuristics for the multiprocessor task scheduling problem. The Genetic Algorithm and Simulated Annealing have been considered in the parameter optimization using statistical techniques for bi-criteria multiprocessor task scheduling represented by DAG.