CHAPTER 5

PROPOSED TASK SCHEDULING ALGORITHMS USING GENETIC APPROACH

Genetic algorithms have been successfully applied to solve a wide range of scheduling problems such as job shop scheduling, time-table problems, task scheduling and allocation onto homogeneous multiprocessor systems and task scheduling and matching in heterogeneous computing environments [37]. Though the task scheduling problem is widely explored for the homogenous processors, only limited work has been carried out for the heterogeneous processors with and without task duplication mechanisms. In this research, the problem of task scheduling using genetic approach is explored and two genetic algorithms have been proposed and developed for completely connected DHCS consisting of bounded number of processors namely, (i) Genetic Algorithm for Task Scheduling (GATS) and (ii) Duplication-based Genetic Algorithm for Task Scheduling (DGATS). The GATS algorithm combines the search capability of genetic algorithms with a list-scheduling heuristic and the DGATS algorithm combines the search capability of genetic algorithms with task duplication-based scheduling heuristic to provide the best-possible solution to a task scheduling problem in an efficient manner when compared to other probabilistic techniques. The proposed algorithms are exemplified with illustration. The correctness and the effectiveness of the GATS and the DGATS algorithms are evaluated by comparing them with the PSGA, HPDCS, HEFT and CPOP algorithms and presented in this chapter.

5.1 Preamble

GAs are the most widely used guided random search techniques for task scheduling problem. GAs are one of the categories of task scheduling algorithms, which provide better results than the other approaches. In contrast to the heuristic scheduling techniques, which require direct information about the application DAG and the heterogeneous computing systems to decide the next scheduling step.
GA-based scheduling techniques operate on chromosomes that represent possible candidate task schedules or the priority of the tasks in the DAG. The general idea of GAs is introduced in this section in order to provide a background for the description of the proposed GA-based algorithms.

The first step necessary to employ a GA is to encode any possible solution to the optimization problem as a set of strings (chromosomes). Each chromosome represents one solution to the problem and a set of chromosomes is referred to as population. The next step is to derive an initial population. A random set of chromosomes is often used as the initial population. Some specified chromosomes can also be included. This initial population is the first generation from which the evolution starts. The third step is to evaluate the quality of each chromosome. Each chromosome is associated with a fitness value, which is, in the case of task scheduling, the completion time of the application. Thus, a chromosome with a smaller fitness value represents a better solution. The objective of the GA search is to find a chromosome that has the optimal (smallest) fitness value.

The selection process is the next step and in this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. Selection is followed by the crossover step. With some probability, some pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in the population may be mutated with some probability. The mutation process transforms a chromosome into another valid one that may or may not already be in the current population. The new population is then evaluated. If the stopping criteria have not been met, the new population goes through another cycle (iteration) of selection, crossover, mutation and evaluation. These cycles continue until one of the stopping criteria is met.

In general, the following are the steps that are taken to implement a GA for a given optimization problem: (1) an encoding, (2) an initial population, (3) an evaluation using particular fitness function, (4) a selection mechanism, (5) crossover.
mechanism, (6) a mutation mechanism and (7) set of stopping criteria. These steps of a typical GA are recounted and presented in Figure 5.1.

```
GA_Based_Task_Scheduling:
begin
  initial population generation;
  evaluation;
  while (stopping criteria not met) do
    begin
      selection;
      crossover;
      mutation;
      evaluation;
    end:
  output best solution;
end.
```

Figure 5.1 Steps in a typical genetic algorithm

The researchers in the past have developed various GA-based task scheduling algorithms for distributed system consisting of homogenous processors [41, 100, 101] and the heterogeneous processors [37, 102, 103]. From the limited literature survey on GAs, it is found that only a very few algorithms have been developed using the genetic approach combined with task duplication. The algorithms also share one common assumption that they were developed for homogeneous processors. The development of scheduling algorithms for heterogeneous processors using genetic approach combined with task duplication for heterogeneous processors has not been explored. For example, the PSGA algorithm [37] which is widely referred to by the researchers uses the earliest finish time heuristic and the earliest start time heuristic to generate the schedules. The major drawbacks of the PSGA algorithm is that it does not consider the possibility of scheduling a task between two already scheduled tasks on the same processor even if a sufficient time slot is available and the task duplication to reduce completion time of the application.
In this research work two genetic algorithms namely, GATS and DGATS have been developed by considering the insertion scheduling policy and the task duplication heuristics. The proposed algorithms are described in the subsequent sections. Finally, the performance comparison of the algorithms is discussed by comparing the algorithms with the PSGA, HEFT, CPOP and the HPDCS algorithms. The other algorithms such as GSA [35] and GA [105] are not considered for comparison, because the chosen algorithm PGSA is already proved to be better than these algorithms.

5.2 Genetic Algorithm for Task Scheduling (GATS)

The GATS consists of two phases. The first phase (steps 1-11) is called an initialization phase and the second phase (steps 12-26) is the iteration phase. Initialization includes reading the DAG, computation cost matrix, number of processors and data transfer rate matrix. The user also provides the number of population (Np), probability of crossover (Xr) and the probability of mutation (Mx). Then the given task graph is level sorted. The Successor Communication Cost (SCC) and the Predecessor Communication Cost (PCC) of each task in the DAG is calculated. The SCC value of a task \( v_i \) is the maximum amount of data to be transferred between task \( v_i \) and all its successor tasks \( v_j \)'s. The SCC value for the exit task is zero since it has no successors. The PCC of a task \( v_i \) is the maximum amount of data to be transferred between task \( v_i \) and its predecessor tasks \( v_k \)'s. The PCC value of entry task is zero, since it has no immediate predecessors. The algorithm then generates the initial population and schedules using the Levelized Earliest Finish Time (LEFT) decoding heuristic.

In the second phase, iteration is carried out for the selection of parent chromosomes for reproduction using select function. The crossover function and mutate function are called to obtain offspring from the selected parents for the new generation. Then this phase involves the assignment of the new population as the current population, applications of LEFT decoding heuristic to obtain solutions and the evaluate function to calculate the fitness of each chromosome's solution. The iteration terminates when the stopping criterion is met. Finally, the best schedule is
reported. The GATS algorithm is exemplified using the task graph given in Figure 5.2 and its computation costs given in Table 5.1. The pseudo code of the proposed GATS algorithm is given in Figure 5.3.

![Task Graph](image)

**Figure 5.2 A random task graph with 15 tasks**

| Table 5.1 Computation costs of the tasks in Figure 5.2 on three processors |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 | v9 | v10 | v11 | v12 | v13 | v14 | v15 |
| p1 | 31 | 40 | 78 | 64 | 37 | 58 | 22 | 30 | 10 | 55 | 24 | 83 | 12 | 36 | 96 |
| p2 | 47 | 69 | 53 | 75 | 81 | 21 | 46 | 13 | 91 | 10 | 37 | 95 | 50 | 74 | 21 |
| p3 | 24 | 94 | 62 | 45 | 61 | 88 | 16 | 68 | 62 | 59 | 48 | 41 | 91 | 20 | 48 |
GATS Algorithm

Input:
- Number of tasks $v$ and the processors $p$
- Computation cost matrix of the DAG: $T(v \times v)$
- Amount of data to be transferred between the tasks: $D(v \times v)$
- Rate of data transfer between the processors: $R(p \times p)$
- Number of population ($N_p$), Probability of crossover ($X_r$) and Mutation ($M_r$)

Output:
- Minimum makespan with less number of processors

1. read the DAG, computation cost matrix, number of processors, the rate of data transfer between the processors matrix $R$, $N_p$, $X_r$ and $M_r$

/* Initialization phase */
2. level sort the given DAG;
3. for each task $v_i$ in level $L_i$
4. begin
5. computer SCC($v_i$) = max[$Data(v_i,v_j)$], where $v_j \in succ(v_i)$;
6. compute PCC($v_i$) = max [$Data(v_i,v_k)$], where $v_k \in pred(v_i)$;
7. end;
8. generate initial_population ( $p_{initial}$ );
9. $p_{current} \leftarrow p_{initial}$. /* copy initial population to current population */
10. schedules $\leftarrow LEFT\_decoding\_heuristic ( p_{current} )$;
11. best_schedule $\leftarrow evaluate (schedules)$;

/* Iteration phase */
12. while stopping criterion not satisfied do
13. begin
14. $p_{new} \leftarrow \{ \}$; /* empty new population */
15. i =1
16. repeat for $N_p$/2 times
17. begin
18. dad $\leftarrow select ( p_{current}, sum\_of\_fitness )$;
19. mom $\leftarrow select ( p_{current}, sum\_of\_fitness )$;
20. $P_{new} \leftarrow P_{new} \cup \text{crossover} (\text{dad, mom, child1, child2, } Xr);$  
21. \hspace{1em} end;  
22. \hspace{1em} until i = Np/2;  
23. \hspace{1em} for each chromosome $\in P_{new}$ do  
24. \hspace{2em} mutate (chromosome, $M_r$);  
25. \hspace{1em} $p_{current} \leftarrow p_{new};$  
26. \hspace{1em} schedules $\leftarrow LEFT_{decoding\_heuristic} (p_{current});$  
27. \hspace{1em} best_schedules $\leftarrow evaluate (schedules);$  
28. \hspace{1em} end;  
29. report the best schedule.

Figure 5.3 Pseudo code of the GATS Algorithm

5.2.1 Chromosome Representation and Initial Population

A chromosome in traditional GAs is often a string that represents a valid solution to the problem to be solved, whereas a chromosome in the GATS represents some attribute of the problem data such as the priority of the tasks in the DAG, which is the key difference between standard GAs and the GATS algorithm in the encoding of chromosomes. This chromosome information is used by a LEFT scheduling heuristic to generate a solution to the problem.

In the initial population generation step, a predefined number (number of tasks in the task graph) of chromosomes are generated and the collection of these chromosomes forms the initial population. As each chromosome contains genes, in the case of task scheduling problem, a gene $i$ in the chromosome represents the priority of the corresponding task $v_i$. For the first chromosome the priority of a task $v_1$ is its SCC value. The SCC for task $v_1$ of the example DAG shown in Figure 5.2 is 54, i.e., for the task $v_1$, the successor tasks are $v_2$, $v_3$, $v_4$, $v_5$ and $v_6$ respectively. Correspondingly the amount of data to be transferred from $v_1$ to $v_2$, $v_3$, $v_4$, $v_5$ and $v_6$ are 36, 49, 54, 41 and 49 respectively. Among these, 54 is the highest value and hence it is the SCC value for task $v_1$. The SCC value for task $v_2$ is 65, because the amount of data
to be transferred from task $v_2$ to $v_7$ is 65 which is greater than the amount of data to be transferred from task $v_2$ to $v_k$ which is 32 only.

Another parameter called $PCC$ for each task is also calculated. The PCC value for task $v_1$ is zero since it has no predecessor. The PCC value for tasks $v_2$, $v_4$, $v_5$ and $v_6$ are 36, 49, 54, 41 and 49 respectively since these tasks have only one predecessor task (task $v_1$) and 36, 49, 54, 41 and 49 are the amount of data to be transferred from task $v_1$ to $v_2$, $v_4$, $v_5$ and $v_6$ respectively. The $PCC$ value for task $v_7$ is 65 (i.e., task $v_7$ has two predecessors task $v_2$ and $v_5$ and the amount of data to be transferred between $v_2$ to $v_7$ and $v_5$ to $v_7$ are 65 and 51 respectively, and among this value 65 is the maximum and hence it is the $PCC$ value for task $v_7$). The $SCC$ of task $v_{15}$ is 0 because it has no successors. Similarly the $SCC$ and $PCC$ values are computed for all the tasks in the task graph in Figure 5.2 and are shown in Table 5.2.

Table 5.2 $PCC$ and $SCC$ values computed for the task graph shown in Figure 5.2

<table>
<thead>
<tr>
<th>Task</th>
<th>Level</th>
<th>SCC</th>
<th>$PCC$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>65</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>69</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>81</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>47</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>85</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>44</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>35</td>
<td>81</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>31</td>
<td>85</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>92</td>
<td>44</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>24</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>33</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>0</td>
<td>33</td>
</tr>
</tbody>
</table>

As mentioned earlier, the gene value (i.e., priority) of a task $v_i$ in the first chromosome is set to the $SCC$ of a task $v_i$. The gene values of the rest of the chromosomes (chromosome 2 to $N_p$) in the initial population are generated...
by a random perturbation of the gene values of the first chromosome as given below:

\[(GT_i') \parallel (GT_i') \parallel \text{Uniform}(PCC/2, -PCC/2)\]  \hspace{1cm} (5.1)

where \((GT_i')\) is the priority of a task \(v_i\) in the first chromosome, \(i = 1, \ldots, n\). \text{Uniform}(PCC/2, -PCC/2) is a random number generated uniformly between \(+ (PCC/2)\) and \(- (PCC/2), j = 2, \ldots, Np\). To reduce the complexity of the GATS algorithm, the number of chromosomes in the populations or the population size, is fixed throughout its operation.

Using the above rules, an initial population of size fifteen for the DAG of Figure 5.2 is generated and shown in Table 5.3. Here, each chromosome has a different priority value for each task in different chromosomes. So each chromosome guides the heuristic to generate a different solution to the problem, which in this case will be a different schedule. Since the \(PCC\) of task \(v_i\) in the task graph shown in Figure 5.2 is 0, there is no perturbation in the priorities of these tasks. Thus, the priority (gene value) of this task for the example DAG remains the same for all the chromosomes in the initial population.

5.2.2 Chromosome Decoding

GATS algorithm uses LEFT as the decoding heuristic to generate the schedules. In this heuristic, the tasks are sorted by level numbers in such a way that all the tasks of a particular level are grouped together and a task ready list, in which the tasks are arranged in a descending order according to their priorities in the chromosome, is built. If there are two tasks with the same priority, then the tie will be resolved by selecting a task based on the breadth first search order. The task \(v_i\) from the ready list with the highest priority is selected and scheduled to the most suitable processor \(p_i\) which gives the minimum \(EFT\) value using the insertion-based scheduling policy. Task \(v_i\) is then deleted from the ready list. The heuristic repeats the matching and scheduling with updates to the ready list, until the list becomes empty and all the tasks are scheduled. Pseudo code for the decoding heuristic is given in Figure 5.4.
Table 5.3 Initial population of fifteen chromosomes generated for the task graph shown in Figure 5.2

<table>
<thead>
<tr>
<th>Task No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 1</td>
<td>54</td>
<td>65</td>
<td>69</td>
<td>81</td>
<td>47</td>
<td>85</td>
<td>44</td>
<td>35</td>
<td>31</td>
<td>92</td>
<td>24</td>
<td>15</td>
<td>33</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>Chromosome 2</td>
<td>54</td>
<td>70</td>
<td>63</td>
<td>97</td>
<td>20</td>
<td>95</td>
<td>103</td>
<td>92</td>
<td>64</td>
<td>82</td>
<td>23</td>
<td>-15</td>
<td>118</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Chromosome 3</td>
<td>54</td>
<td>54</td>
<td>21</td>
<td>106</td>
<td>58</td>
<td>61</td>
<td>85</td>
<td>107</td>
<td>100</td>
<td>78</td>
<td>2</td>
<td>28</td>
<td>-5</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>Chromosome 4</td>
<td>54</td>
<td>91</td>
<td>21</td>
<td>74</td>
<td>84</td>
<td>107</td>
<td>77</td>
<td>41</td>
<td>-32</td>
<td>107</td>
<td>38</td>
<td>31</td>
<td>83</td>
<td>39</td>
<td>18</td>
</tr>
<tr>
<td>Chromosome 5</td>
<td>54</td>
<td>31</td>
<td>71</td>
<td>96</td>
<td>78</td>
<td>85</td>
<td>20</td>
<td>19</td>
<td>14</td>
<td>75</td>
<td>53</td>
<td>34</td>
<td>-54</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>Chromosome 6</td>
<td>54</td>
<td>40</td>
<td>59</td>
<td>124</td>
<td>46</td>
<td>102</td>
<td>24</td>
<td>112</td>
<td>88</td>
<td>72</td>
<td>-5</td>
<td>34</td>
<td>51</td>
<td>8</td>
<td>-2</td>
</tr>
<tr>
<td>Chromosome 7</td>
<td>54</td>
<td>43</td>
<td>109</td>
<td>41</td>
<td>11</td>
<td>102</td>
<td>-11</td>
<td>56</td>
<td>107</td>
<td>65</td>
<td>37</td>
<td>21</td>
<td>52</td>
<td>28</td>
<td>-11</td>
</tr>
<tr>
<td>Chromosome 8</td>
<td>54</td>
<td>65</td>
<td>100</td>
<td>133</td>
<td>84</td>
<td>51</td>
<td>27</td>
<td>50</td>
<td>85</td>
<td>122</td>
<td>42</td>
<td>3</td>
<td>-49</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>Chromosome 9</td>
<td>54</td>
<td>62</td>
<td>97</td>
<td>108</td>
<td>43</td>
<td>69</td>
<td>-3</td>
<td>-22</td>
<td>16</td>
<td>121</td>
<td>55</td>
<td>37</td>
<td>63</td>
<td>37</td>
<td>2</td>
</tr>
<tr>
<td>Chromosome 10</td>
<td>54</td>
<td>97</td>
<td>80</td>
<td>33</td>
<td>13</td>
<td>59</td>
<td>82</td>
<td>29</td>
<td>-32</td>
<td>98</td>
<td>49</td>
<td>31</td>
<td>-7</td>
<td>27</td>
<td>-16</td>
</tr>
<tr>
<td>Chromosome 11</td>
<td>54</td>
<td>68</td>
<td>110</td>
<td>62</td>
<td>86</td>
<td>44</td>
<td>89</td>
<td>47</td>
<td>80</td>
<td>110</td>
<td>-11</td>
<td>29</td>
<td>-44</td>
<td>31</td>
<td>18</td>
</tr>
<tr>
<td>Chromosome 12</td>
<td>54</td>
<td>78</td>
<td>57</td>
<td>132</td>
<td>57</td>
<td>85</td>
<td>48</td>
<td>87</td>
<td>14</td>
<td>69</td>
<td>27</td>
<td>20</td>
<td>47</td>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td>Chromosome 13</td>
<td>54</td>
<td>98</td>
<td>82</td>
<td>63</td>
<td>43</td>
<td>80</td>
<td>74</td>
<td>30</td>
<td>15</td>
<td>128</td>
<td>16</td>
<td>18</td>
<td>-16</td>
<td>19</td>
<td>-18</td>
</tr>
<tr>
<td>Chromosome 14</td>
<td>54</td>
<td>31</td>
<td>66</td>
<td>58</td>
<td>10</td>
<td>100</td>
<td>79</td>
<td>14</td>
<td>-3</td>
<td>116</td>
<td>18</td>
<td>-2</td>
<td>-47</td>
<td>33</td>
<td>15</td>
</tr>
<tr>
<td>Chromosome 15</td>
<td>54</td>
<td>82</td>
<td>59</td>
<td>93</td>
<td>64</td>
<td>76</td>
<td>-3</td>
<td>99</td>
<td>73</td>
<td>80</td>
<td>42</td>
<td>18</td>
<td>-24</td>
<td>23</td>
<td>12</td>
</tr>
</tbody>
</table>
LEFT_Decoding_Heuristic

1. begin
2. for each level $L_i$ of the DAG do
3. begin
4. ready-list $\leftarrow$ list of tasks in level $L_i$ according to the decreasing order of priority from the chromosome;
5. while ready-list $\not\leftrightarrow$ empty do
6. begin
7. select the highest priority task $v_i$ from the ready-list;
8. compute EFT of task $v_i$ on all the processors in the processor set $P$ using the insertion-based scheduling policy;
9. assign task $v_i$ to the processor $p_j$ that gives the minimum $EFT$;
10. remove the task $v_i$ from the ready-list;
11. end;
12. end;
13. end.

Figure 5.4 Pseudo code of the LEFT heuristic

5.2.3 Fitness Evaluation

The GATS algorithm uses the schedule length as the objective function for evaluating the schedules corresponding to each chromosome. Since, its value may vary from problem to problem, to maintain uniformity over various problem domains, the fitness to normalize the objective function values to a convenient range of 0 to 1 is used. The following objective-to-fitness mapping [37] function is used:

$$f(i) = \frac{SL(M) - SL(i)}{\sum_{j=1}^{Np} [SL(M) - SL(j)]}$$  \hspace{1cm} (5.2)$$

where $f(i)$ is the fitness of chromosome $i$: $SL(M)$ is the maximum schedule length of a solution corresponding to a chromosome in the current population, $SL(i)$ is the schedule length corresponding to chromosome $i$ and $Np$ is the population size.
5.2.4 Selection and Elitism

GATS combines the exploitation of the past results by selecting parent chromosomes for reproduction based on their fitness with the exploration of new areas in the search space via crossover and mutation. Chromosomes with higher fitness values have a higher probability of contributing one or more offspring in the next generation. This method is a simulated version of the natural selection, a Darwinian survival of the fittest theory [97]. A biased roulette wheel, where each chromosome in the population has a slot sized in proportion to its fitness, performs the selection method. Each time when an offspring is required, a simple spin of the weighted roulette wheel yields a parent chromosome. In this method, a random number \( u \) is generated between 0 and the sum of the fitness. A population member \( j \) whose running sum of the fitness is greater than or equal to \( u \) is selected as a candidate for reproduction. GA-based techniques use many other selection methods such as tournament selection, simulated-annealing selection, etc., but a simple roulette-wheel selection method works well in GATS.

The GATS algorithm also incorporates elitism. The best chromosome is compared with an elite chromosome, a copy of which is stored separately from the population at the end of each iteration. If the best chromosome is better than the elite chromosome, a copy of it becomes the elite chromosome. If the best chromosome is not as good as the elite chromosome, a copy of the elite chromosome replaces the worst chromosome in the population. Elitism is important because it guarantees that the quality of the best solutions found over generations is monotonically increasing.

5.2.5 Swap Crossover

In natural genetics, at the cellular level, a pair of chromosomes strike into one another, exchange chunks of genetic information and drift apart. In GA, this is generally referred to as crossover because of the way in which the genetic material crosses over from one chromosome to another. Crossover incorporates attributes of two parents into a new individual. GATS algorithm apply a 2-point crossover operator to the priority of the chromosome. Since a
chromosome is based on the problem data and has to be decoded using an underlying heuristic, it always generates a feasible solution. The crossover is applied with a certain crossover rate \( Xr \) which is the ratio of the number of offspring produced by crossover in each generation to the population size. It controls the amount of crossover being applied. In a 2-point crossover operator, two cross sites are selected randomly and the values of the priorities between the cross sites are swapped among the two mating chromosomes. For example, consider the two chromosomes (chromosome 1 and chromosome 2) shown in Figure 5.5(a). Two crossover points are located on each chromosome (cross points after task \( v_d \) and before task \( v_{/2} \)). The offspring chromosomes (child 1 and child 2) generated by a crossover operation are shown in Figure 5.5(b).

![Figure 5.5](image)

(a) Task No. → 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Chromosome 1
Chromosome 2

(b) Task No. → 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Child 1
Child 2

Figure 5.5 The swap crossover operator (a) two chosen parent chromosomes and (b) the offspring chromosome created by the parent chromosome in Figure (a)

5.2.6 Swap Mutation

Mutation is a background operator that is used for finding new points in the search space so that population diversity can be maintained. Mutation is done by selecting a gene at random with a probability \( Mr \) and perturbing its value in the range \((-SCC/2)\) to \((SCC/2)\), where SCC is the successor cost of the selected task in the DAG. After perturbation, if the priority value becomes more than \( PCC + SCC \) of the task, then the priority is assigned the value \( PCC + SCC \). If the priority value becomes less than \( PCC \), it is assigned...
the value $PCC$. The objective is to explore a wider space of priorities, but within the proximity of the original problem. For example, in the chromosome 2, for task $v_4$, the values of SCC and the PCC are 81 and 54 respectively. The priority value 97 perturbed with a value lies in the range (-81/2 and 81/2). Say the value is 25, then, the priority value is $97 + 25 = 122$. But its SCC + PCC value $(81 + 54 = 135)$ is greater than 122 and less than its PCC value 54. Hence, the priority of task $v_4$ becomes its PCC value i.e., 54. Thus, after the mutation task $v_4$ of the chromosome 2 becomes 54. This mutation operator is shown in Figure 5.6.

<table>
<thead>
<tr>
<th>Task No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>54</td>
<td>70</td>
<td>63</td>
<td>20</td>
<td>95</td>
<td>103</td>
<td>92</td>
<td>64</td>
<td>82</td>
<td>23</td>
<td>-15</td>
<td>118</td>
<td>25</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Task No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>54</td>
<td>70</td>
<td>63</td>
<td>20</td>
<td>95</td>
<td>103</td>
<td>92</td>
<td>64</td>
<td>82</td>
<td>23</td>
<td>-15</td>
<td>118</td>
<td>25</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

(b)

Figure 5.6 The mutation operator (a) before mutation and (b) after mutation

5.2.7 Termination Criteria

Although static scheduling takes place offline, a limit must be placed on the running time of the scheduling algorithm to provide a practical solution. The maximum running time of the scheduling algorithm depends on the characteristics of the distributed application. The GATS algorithm runs for a predetermined number of generations ($N_g$) which is set to the product of number of depth and breadth of the task graph.

5.3 Duplication-based Genetic Algorithm for Task Scheduling (DGATS)

The DGATS algorithm is similar to the GATS algorithm and the only difference between these two algorithms is that the DGATS algorithm uses the task duplication mechanism. This heuristic schedules the tasks of the task graph in
a level by level manner. i.e., all the tasks at a lower level are scheduled first before scheduling the tasks at the higher level. Initially the highest priority task is selected at a level and scheduled to the suitable processors which minimize the EFT of the task. Here, task duplication is applied, if sufficient free slot is available to duplicate the predecessor task and only when the duplication reduces the EST of the task. By applying the LEFT heuristic to each chromosome in the population, different schedules can be generated corresponding to each chromosome. The pseudo code of the DLEFT heuristic is given in Figure 5.7.

DLEFT Heuristic

1. begin
2. for each level Li of the DAG do
3. begin
4. ready-list ← list of tasks in level Li according to the decreasing order of the priority from the chromosome;
5. while ready-list <> empty do
6. begin
7. select the highest priority task vi from the ready-list;
8. s1 ← EFT of task vi on all the processors in the processor set P using insertion-based scheduling policy and without task duplication;
9. s2 ← EFT of task vi on all the processors in the processor set P using insertion-based scheduling policy and with task duplication;
10. if s1 less than s2 then
11. assign task vi to processor pj;
12. undo task duplication;
13. else
14. assign task vi to processor pj;
15. end;
16. end;
17. end.

Figure 5.7 Pseudo code of the DLEFT heuristic
For illustration, the best SL generated by the DGATS algorithm for the task graph given in Figure 5.2 is 350 and is equal to the SL generated by the HPDCS and lesser than the SL generated by the GATS, PSGA, HEFT and the CPOP algorithms. The SL generated by the HPDCS, GATS, PSGA, HEFT and the CPOP algorithms are 350, 368, 383, 405 and 426 respectively.

5.4 Experimental Results and Discussion

This section presents the comparative evaluation of proposed algorithms GATS and DGATS with the existing algorithms such as the CPOP, HEFT, HPDCS and the PSGA. Two sets of graphs, randomly generated task graphs and the graphs that represent some of numerical real world problems such as Gaussian elimination algorithm, Fast Fourier transformation, Cholesky factorization graphs and the diamond graphs are considered as the workload for testing the algorithms. Experiments were conducted using Intel Pentium 4 processors (3.06 GHz, 512 MB RAM) with different task graphs structures and are organized in three test suites. For all the test suites, the value of \( X_r \) and \( M_r \) are fixed with 0.6 and 0.05 respectively, for the comparison purpose with the existing algorithms. In the test suite 1 and test suite 2, the performance of the proposed algorithms are evaluated for a large set of randomly generated task graphs and in test suite 3, the performance of the proposed algorithms are compared using the application graphs.

Test Suite 1: Two types of experiments are conducted in this test suite. First by varying the task graph with task size varying from 20 to 100 with step size of 10 and scheduling this task graphs on the DHCS consisting of five processors. The result obtained from this experiment is shown in Figure 5.8. Each data point in the figure is the average of 60 experiments. The schedule length based performance ranking of the algorithms is DGATS, HPDCS, GATS, PSGA, HEFT and CPOP. The ranking starts with the best algorithm and ends with the worst. On an average the DGATS algorithm is better than the CPOP algorithm by 30 percent, the HEFT algorithm by 23 percent, the PSGA algorithm by 17 percent, the GATS algorithm by 11 percent, and the HPDCS algorithm by 7 percent with respect to average schedule length for the random task graphs.
In the second experiments, the performances of the algorithms are evaluated by varying the number of processors in the heterogeneous computing system. For this experiment, around 250 random task graphs with 120 tasks each (with different graph characteristics) have been generated and scheduled on to a DHCS consisting of processors varying from 4 to 20 incrementing by step size of 4. The results obtained by this experiment are shown in Figure 5.9 and it shows that the proposed algorithm DGATS surpasses the existing algorithms.

![Figure 5.8 Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the varying number of tasks](image)

![Figure 5.9 Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the varying number of processors](image)
Test Suite 2: In this test suite, the quality of schedules generated by the algorithms with respect to CCR values and graph structure was compared. In the first experiment, task graphs with CCR value ranging from 0.1, 0.5, 1.0, 5.0, and 10.0 are scheduled on a heterogeneous computing system consisting of five processors and the result obtained from this experiment is shown in Figure 5.10. Each data point in the figure is the average of 60 experiments. The performance ranking of the algorithms is DGATS, HPDCS, GATS, PSGA, HEFT and CPOP. The ranking starts with the best algorithm and ends with the worst.

In the second experiment the quality of schedules generated by the algorithms with respect to the graph structure is evaluated. A dense graph (shorter graph with high parallelism) and a longer graph (low parallelism) can be generated by selecting $\alpha >>1.0$ and $\alpha <<1.0$ respectively. The performance of the algorithms are evaluated by average schedule length with respect to the graph structure, by varying the value $\alpha$ with 0.5, 1.0 and 2.0 and the results obtained by this experiment are shown in Figure 5.11. Each data point in the figure is the average of 75 experiments.

![Figure 5.10 Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the different CCR values](image-url)
Figure 5.11 Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the different graph structures

Test Suite 3: In this test suite, the performances of the algorithms are evaluated using the application graphs of four real world problems such as Gaussian Elimination algorithm, Fast Fourier Transformation, Cholesky factorization graphs and diamond graphs. The miniature examples of Gaussian Elimination graphs, Fast Fourier Transformation graphs are shown in Figure 3.11(b) and 3.15(b) respectively. To evaluate the performance of the algorithms using Gaussian Elimination application, the CCR and range percentage values given in section 3.4.2 are alone used since the structure of the application is known. The other parameter like the number of tasks and the out degree is not used. The performances of the algorithms are evaluated at various matrix sizes from 5 to 15 with an increment of one. The smallest size of the task graph in this experiment has 14 tasks and the largest one has 119 tasks. The simulation results are given in Figure 5.12(a).

For the efficiency comparison, the number of processors used in our experiments is varied from 2 to 12. The CCR and range percentage parameter values are the same. Figure 5.12(b) gives the efficiency comparison for Gaussian elimination graphs when the matrix size is 15. Each data point in the figure is the average of 50 experiments. The experiments conducted using GE application graphs show that the DGATS algorithm outperforms HPDCS, GATS, PSGA, HEFT and CPOP algorithms in terms of schedule length and efficiency.
Figure 5.12 Performance of DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the GE graphs in terms of (a) average SL and (b) efficiency.

For the experiments of Fast Fourier Transformations, the CCR and range percentage values given in section 3.4.2 were used. The number of data points in the experiments varies from 2 to 32 incrementing powers of 2. The smallest size graph in this experiment has 5 tasks and the largest one has 223 tasks. The simulation results are given in Figure 5.13(a).
For the efficiency comparison, the number of processors used in the experiments varies from 2 to 32, incrementing by the power of 2. The CCR and range percentage parameter value are the same. Figure 5.13(b) gives the efficiency comparison for Gaussian elimination graphs when the matrix size is 32. Each data point in the figure is the average of 50 experiments. The experiments carried out using the FFT graphs also confirm that the proposed algorithm DGATS outperforms all the existing algorithms and the GATS algorithm shows better results than the HEFT, CPOP and PSGA in terms of both average schedule length and efficiency.

![Graph (a)](image)

![Graph (b)](image)

Figure 5.13 Performance of DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the FFT graphs in terms of (a) average SL and (b) efficiency
In another experiment, the performance of the GATS and DGATS have been compared with the CPOP, HEFT, HPDCS and the PSGA algorithm using special classes of DAGs, such as the diamond DAGs and the task graphs of Cholesky factorization. The diamond graphs and the Cholesky factorization are shown in Figure 5.14 and Figure 5.15 respectively.

The diamond DAGs are similar to the master-slave DAGs, in which the master gives instructions to the slaves and the slaves send the result back to the
master node. A diamond DAG is characterized by the width \( n \) of the DAG or the number of slave tasks at each level. The experiments were conducted by generating the diamond DAGs with the value of \( n \) ranging from 3 to 10 in step size of one. The DAGs are scheduled onto the DHCS consisting of processors ranging from 3 to 10. The result obtained by this experiment is shown in Figure 5.16. Each data point in the figure is the average of 25 experiments. The simulation results shows that the DGATS algorithm outperforms the existing algorithms CPOP, HEFT, PSGA, GATS, and HPDCS particularly when the dimension of the matrices is greater than five. The GATS and the PSGA algorithms uniformly give similar schedule length.

![Diagram](image)

Figure 5.16 Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the Diamond graphs

Cholesky factorization applications operate on matrix, the number of nodes and the edges in this task graph depends on the size of the data matrix. The number of nodes in the task graph for each application is \( (n^2+3n)/2 \), where \( n \) is the dimension of the matrix. A special case of Cholesky factorization graph for matrix dimension equal to 4 is given in Figure 5.15. In the experiment of Cholesky factorization, the value of \( n \) is varied from 6 to 10 with increments of three so that the numbers of nodes in the graphs range from 27 to 234. These graphs are scheduled on the DHCS consisting of processors ranging from 3 to 15 in step size of 3. The results obtained from this experiment are plotted and shown in the Figure 5.17. The experiments conducted using Cholesky factorization graph shows
that the DGATS algorithm significantly outperforms the CPOP, HEFT, PSGA, GATS and the HPDCS algorithms.

![Graph showing average schedule length](chart.png)

**Figure 5.17** Average SL generated by DGATS, GATS, HPDCS, PSGA, HEFT and CPOP algorithms for the Cholesky factorization graphs

### 5.5 Summary

Genetic algorithms have been effectively applied to solve a wide range of scheduling problems. GAs provide better schedules than the other task scheduling approaches such as list-based and clustering and they are generally used where an optimal schedule is needed. In this chapter two task scheduling algorithms namely, GATS and DGATS have been designed and developed using genetic approach. The performances of the algorithms are compared with the well known and widely referred task scheduling algorithms such as PSGA, HEFT and CPOP using a large set of randomly generated task graphs and graphs of some real world numerical problems by simulations. The experimental results show that the DGATS outperforms the existing algorithms such as CPOP, HEFT, PSGA and HPDCS. The GATS algorithm gives considerable improvement over the PSGA algorithm and substantially surpasses HEFT and CPOP algorithms. The HPDCS algorithm gives better results than the GATS algorithm because it uses a task duplication-based approach.