Chapter Six

GRID SCHEDULING STRATEGY USING GENETIC ALGORITHM
6.1 Introduction

The purpose of grid computing is to utilize computational power of geographically distributed resources. The usefulness of a grid system largely depends on the efficiency of the system regarding the allocation of jobs to grid resources. The task scheduling in grid environments strive to maximize the overall performance of the grid, by minimizing the makespan and communication cost while maximizing the resource utilization. This chapter proposes the Grid Scheduling Strategy using Genetic Algorithm\(^1\) (GSSGA) to schedule a group of tasks in a grid environment. Genetic algorithm (GA) is an adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. The proposed methodology uses the genetic algorithm to find an efficient schedule in grid computing and adapts new fitness function to find the suitability of the schedule. Furthermore, the crossover and mutation operations in the algorithm can move the solution away from the local-optimal solution towards a near-optimal solution. The proposed algorithm has been implemented and tested with arbitrary task graphs in a simulated environment. The results show that the proposed GSSGA outperformed all other scheduling algorithms across a range of scenarios. The scheduling tool has been developed to conduct different experiments with arbitrary task graphs for all the proposed algorithms.

\(^1\) This algorithm has been published “Grid scheduling strategy using Genetic Algorithm (GSSGA)”, International Journal of Computer Technology and Applications (ISSN: 2229-6093-Online), Vol. 3, Issue 5, pp.1800-1806, September-October 2012.
namely, LCTSA, EDOS, TDB-EMOS and GSSGA and also used for the well known Min-Min, HEFT and EDS-G algorithms.

6.2 Fundamentals of Genetic Algorithm (GA)

Genetic Algorithm is a family of computational models inspired by evolution. GA is a procedure used to find approximate solutions to search problems through application of the principles of evolutionary biology. Genetic algorithms use biologically inspired techniques such as genetic inheritance, natural selection, mutation and reproduction (recombination or crossover).

The genetic algorithm differs from other search methods. GA searches among population of points and works with a coding of parameter set, rather than the parameter values. It also uses objective function without any gradient information.

GA is a technique to find optimal or nearly optimal solutions of search problems. In 1960, John Holland had an assertive thought and worked on GA. He published the first paper “Adaptation in Natural and Artificial System” in the year 1975 [Gol 89]. He invented GA as meta-heuristic search based on “Survival of the fittest” a common ideology of biology. He introduced mutation and reproduction methodology of biology into the artificial system. From that time the terms Gene, Chromosome, Individual, Population, Crossover and Mutation are used in the search technique.

6.2.1 Chromosome Encoding

A chromosome is a set of parameters which define a proposed solution of a problem. The key issue of genetic algorithm is to encode the solutions of the problem as chromosomes. In order to use GAs to solve the optimization problem, variables $x_i$’s
are first coded in some string structures. The length of the string is usually determined according to the desired solution accuracy [Kal 05]. Various encoding methods have been created for a particular problem to provide effective implementation of genetic algorithm. The encoding methods can be classified as Binary encoding, Real-number encoding, Integer or literal permutation encoding and Value encoding.

6.2.1.1 Binary Encoding

Binary encoding (i.e., the bit strings) is the most common method. The string consists of combinations of 1’s and 0’s. It is easy to change the bit wise operations. Initially this encoding method is used by the early researchers and the majority of theories as well as assumptions are using binary encoding. But the binary encoding is almost impossible to represent the problem of industrial engineering world.

6.2.1.2 Real Number Encoding

Real number encoding is used for function optimization problems. It has been widely confirmed that it performs better than binary encoding for function optimization and constrained optimization problems. It is easy to form effective genetic operators by borrowing useful techniques from conventional methods.

6.2.1.3 Integer or Literal Permutation Encoding

Integer or literal permutation encoding is used for combinational optimization problems because the essence of this kind of problems is to search for the best permutation or combination of items.
6.2.1.4 Value encoding

Direct value encoding can be used in problems where some complicated value such as real numbers are used. Use of binary encoding for this type of problems would be very difficult. Values of solution can be connected to problem which may be integer numbers, real numbers, alphabet characters or combination of all these to form a complicated object. In value encoding, every chromosome is a string of alphanumeric values. Many encoding methods have been proposed for scheduling method [Che 96], in which the value encoding method is well suited.

6.2.2 Fitness Function

Fitness function is used to measure the quality of individuals in the population. The fitness function should encourage the formation of the solution to achieve the objective function. After the fitness evaluation process, the new individuals are compared with the previous generation. The selection process is then conducted to retain the fittest individuals in the population as successive generations. The fitness function quantifies the optimality of a solution so that a particular solution may be ranked against all the other solutions. It depicts the closeness of a given solution to the desired result.

6.2.3 Selection or Reproduction

The selection operator makes more copies of better strings in a new population which has been extracted from an existing population. The new population with good strings forms a mating pool. To sustain the generation of a new population, the reproduction of the individuals in the current population is necessary. For selecting better individuals the selection must be from the fittest individuals of the previous population. Many methods for selecting the fittest individuals have been used for
scheduling such as *Roulette wheel selection, rank selection, tournament selection, steady state* and *elitism*.

**Roulette wheel Selection** is also known as fitness proportionate selection. It is used for selecting potentially useful solutions for recombination. The chance of an individual being selected is proportional to its fitness, greater or lesser than its competitor’s fitness. In Roulette wheel, individuals are selected with a probability that is directly proportional to their fitness values specifically an individual's selection corresponds to a portion of a Roulette wheel. The probabilities of selecting a parent can be seen as spinning a Roulette wheel with the size of the segment for each parent being proportional to its fitness. Obviously, those with the highest fitness (i.e. largest segment sizes) have more probability of being chosen. The fittest individual occupies the largest segment, whereas the least fit have correspondingly smallest segment within the Roulette wheel. The circumference of the Roulette wheel is the sum of all fitness values of the individuals. The Roulette wheel mechanism is depicted in Figure 6.1.

![Figure 6.1 Roulette wheel marked for five individuals according to fitness](image-url)
Figure 6.1 shows a Roulette wheel for five individuals having different fitness values. Since the fifth individual has higher fitness than any other, it is expected that the Roulette wheel selection will choose the fifth individual than any other individual. However, all individuals have a chance, with a probability that is proportional to its width. By repeating this, each time an individual needs to be elected. The better individuals will be selected more often than the poorer ones, thus fulfilling the requirements of survival of the fittest. Let \( f_1, f_2, \ldots, f_n \) be fitness values of individual 1, 2, \ldots, \( n \). Then the selection probability, \( p_i \) for individual \( i \) is define as,

\[
p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}
\]

In order to choose \( n \) strings, the Roulette wheel is spun for \( n \) times. The basic advantage of Roulette wheel selection is that it discards none of the individuals in the population and gives a chance to all of them to be selected. Therefore, diversity in the population is preserved.

**Rank Selection** assigns numerical rank value based on fitness of each individual in the population. In this selection, the individuals in the population are sorted from best to worst according to their ranks. The highest rank individual gets a brighter chance of selection for the next generation. The advantage of this method is that it can prevent least fit individuals from gaining dominance. The disadvantage of this method is that it requires sorting the entire population by rank which is a potentially time consuming procedure.

**Tournament Selection** randomly selects a set of individual and picks out the best individual for reproduction. The number of individual in the set is called the tournament size. The tournament selection strategy provides selection pressure (the
degree to which the better individuals are favored) by holding a tournament competition among \( N_U \) individual (Frequency of \( N_U = 2 \) [Gol 91]. A common tournament size is two, and this is called binary tournament. The best individual (the winner) from the tournament is the one with highest fitness which is the winner of \( N_U \). Tournament competitors and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for new offspring is filled. By adjusting tournament size, the selection procedure can be made arbitrarily large or small.

**Steady state Selection** is that bigger part of chromosome should survive to next generation. In every generation, a few (good individuals with high fitness for maximization problem) chromosomes are selected for creating new offsprings. Then some (bad with low fitness) chromosomes are removed and new offspring is placed in that place. The rest of population survives a new population.

**Elitism** is an addition to many selection methods which forces genetic algorithms to retain some number of the best individual at each generation. It improves the selection process and save the best individuals. With elitist selection, the quality of the best solution in each generation monotonically increases over time. Without elitist selection, it is possible to lose the best individuals due to stochastic errors (due to crossover, mutation or selection procedure). Elitism can be incorporated with two selection methods, namely, rank selection and Roulette wheel selection, by first copying the fittest individual into the next generation and then using the rank selection or Roulette wheel selection to construct the rest of the population. Hou et. al. [Hou 94] showed that the elitism method can improve the performance of the genetic algorithm.
6.2.4 Crossover

After reproduction phase is over, the population is enriched with better individuals. Reproduction makes clones of good strings, but does not create new ones. The aim of the crossover operator is to search the parameter space. It is a recombination operator which proceeds in three steps. First, the reproduction operator selects random pair of two individual strings for mating, then a cross-site is selected at random along the string length and the position values are swapped between two strings following the cross-site. The combined action is responsible for much of GA’s power. There are five types of crossover operators such as single-point crossover, two-point crossover, uniform crossover, arithmetic crossover and heuristic crossover.

**Single-point crossover** randomly selects one crossover point and then copies everything before this point from the first parent and then everything after the crossover point copy from the second parent.

**Two-point crossover** randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

**Uniform crossover** allows the parent chromosomes to be mixed at the gene level rather than at the segment level (as in single and two point crossovers).

**Arithmetic crossover** is producing new offspring from the two parent chromosomes using the generated equations given below:

\[
Offspring_1 = a \times \text{Parent}_1 + (1-a) \times \text{Parent}_2
\]

\[
Offspring_2 = (1-a) \times \text{Parent}_1 + a \times \text{Parent}_2
\]

where \(a\) is a random weighting factor chosen before each crossover operation.
**Heuristic crossover** uses the fitness values of the two parent chromosomes to determine the direction of the search.

\[
\text{Offspring}_1 = \text{Best Parent} + r*(\text{Best Parent}- \text{Worst Parent})
\]

\[
\text{Offspring}_2 = \text{Best Parent}
\]

where \( r \) is a random number between 0 and 1.

Even though more type of crossover operators are available, it is difficult to generalize the optimal crossover operation. The selection of crossover operators is made such that the search in genetic space is proper.

### 6.2.4.1 Crossover Rate

In GA literature, the term crossover rate is usually denoted as \( P_C \), the probability of crossover. The probability varies from 0 to 1. This is calculated in GA by finding out the ratio of the number of pairs to be crossed to some fixed populations. Normally crossover rates are ranged from 0.5 to 1.

### 6.2.5 Mutation

After crossover, the strings are subjected to mutation. Mutation of a bit involves flipping it, changing 0 to 1 and vice versa with a small mutation probability. The bits of the strings are independently muted, that is, the mutation of a bit does not affect the probability of mutation of other bits. The mutation is simply an insurance policy against the irreversible loss of genetic material. It is also used to maintain diversity in the population. If true optimal solution requires at one position with one change, then neither reproduction nor crossover operator will be able to create the change in that position. The mutation is the operator which makes this type of change. Hence, mutation causes movement in the search space (local or global) and restores lost information to the population.
6.2.5.1 Mutation Rate

Mutation rate denoted as $P_M$ is the probability of mutation which is used to calculate number of bits to be muted. The mutation operator preserves the diversity among the population which is also very important for the search. Mutation probabilities are smaller in natural population leading us to conclude that mutation is appropriately considered a secondary mechanism of GA adoption. In general, the mutation rates are varying from 0.001 to 0.5.

6.2.6 Convergence of GA

The good strings in a population set and random information exchange are simple and straightforward. No mathematical proof is available for convergence of GA [Raj 04]. According to Rajeev and Krishnamoorthy [Raj 92], one criteria for convergence may be such that when a fixed percentage of columns and rows in population matrix becomes the same, it can be assumed that convergence is attained. The fixed percentage may be 80% to 85%.

In GA as we proceed with more generations, there may not be much improvement in the population fitness and the best individual may not change for subsequent populations. As the generation progresses, the population gets filled with more fit individuals with only slight deviation from the fitness of best individuals so far found, and the average fitness comes very close to the fitness of the best individuals.

6.2.7 Benefits of GA

Though a number of different optimization techniques may be applied to solve NP-Complete scheduling problems, GA has the following benefits [Rom 98].
Solution Evaluation instead of Construction. The most obvious benefit is that GAs relieve the user of knowing how to construct a solution. In order to implement a standard problem solving technique, the user is required to know how to construct a solution to a problem. There has to be an algorithm that yields a valid solution to the given problem.

The situation is completely different when using Genetic Algorithm. It treats the problem in an encoded form and operates on bit strings. Thus a GA constructs a solution by rearranging bit strings. These construction steps are not defined by the user, but are inherent to the GA. It is the task of the evaluation function to decode and to assess the solution. This means that the user has to provide knowledge on how to evaluate a given problem.

Assessing a solution instead of constructing it also eases the implementation of further constraints on the task schedule. A new constraint simply adds another penalty term to the fitness function. Thus it is accounted for automatically.

Integration of various problems. GA offers a simple way to integrate different search procedures into one optimization process yielding a global optimum instead of local optima for each search. Each of the involved problems is encoded into a number of chromosomes and has its own fitness function. The problems can be solved in one step by constructing an individual that consists of the chromosomes of all involved problems and by deriving an appropriate algebraic conjunction of the fitness function.

6.3 The Proposed Grid Scheduling Strategy using Genetic Algorithm (GSSGA)

In the present study Genetic Algorithm is used to schedule the tasks of DAG. In the literature it is found that many researchers have used GA for scheduling
independent tasks. But in the present study, a DAG model of application with precedence constraints is considered. The objective function of the proposed GSSGA focuses on minimizing the makespan and communication cost while maximizing the utilization of resources. Conventionally makespan alone is considered as criteria to define the fitness function for scheduling problem.

Some genetic algorithms are using more than one objective function as the fitness function. Albert.Y.Zomaya and Yee-Hwei Teh [Alb 01] have used three objective functions, namely, minimizing the execution time, maximizing the processor utilization and a well balanced load to the resources. Kun-Ming Yu, Cheng-Kwan Chen [Kun 08] proposed the fitness function includes the file size $T(i)_{file}$ in Mega Byte of $T(i)$, the output size $T(i)_{result}$, its length $T(i)_{length}$, bandwidth $B(j)$ and processing capacity $P(j)$ of resource j. Rachhpal Singh [Rac 12] proposed a fitness function which contains the makespan and communication time. Vikas Gaba and Anshu Prashar [Vik 12] proposed a fitness function as average waiting time.

The major steps of GA can be understood easily. GA starts with a set of solutions and ends with optimal solution. The following steps of GA are common and it is being followed in almost all GAs.

The procedure for GSSGA is as follows:


2. [Fitness] Evaluate the fitness function $f(i)$ using GSSGA fitness of each chromosome $i$. 

138
3. [New Population] Create a new population by repeating following steps until the new population is complete.

   (i) [Selection] Select the chromosomes from the population calling *GSSGA Selection* to perform GA operations.

   (ii) [Crossover] Crossover the parents to form new offspring using *GSSGA Crossover*.

   (iii) [Validity] The new offspring has to be checked for its validity.

   (iv) [Mutation] Mutate new offspring at each position in chromosome *GSSGA Mutation*, if needed.

4. [Accepting] The valid chromosomes are used in the process of producing new generation.

5. [Test] If the end condition is satisfied, stop and return the best solution from the current generation

6. [Loop] Otherwise repeat the same process from step 2.

### 6.3.1 Chromosomal representation of GSSGA

The first phase of the Genetic Algorithm is the proper representation of chromosome. The chromosome representation is the parameters available in the solution. The value encoding method is suitable for scheduling problem [Che 96]. The solution consists of task number and resource identification number (id) in a sequence. The task number and resource id are denoted as integer number. For example, the set (4, 3) is the 4\textsuperscript{th} task mapped with 3\textsuperscript{rd} resource. The chromosome may be called as string in this chapter. The examples of encoded chromosomes relating to this problem are:
Chromosome 1 = (1,4),(3,2),(3,1),…(n, m)
Chromosome 2 = (1,2),(4,1),(5,3),…(n, m)

where \( n \) number of tasks and \( m \) number of resources. The valid scheduled solutions alone are considered for the chromosome. These chromosomes are used for the initial population.

### 6.3.2 Initial Population of GSSGA

The size of the population of chromosomes is to be 20. In which the four chromosomes are generated through Min-Min, HEFT, LCTSA and EDOS algorithms and the rest are generated randomly with task number and resources id. The initial population of GSSGA is generated using the procedure GSSGA population as follows:

**GSSGA Population**

1. Perform schedule using existing scheduling algorithms
2. Get the solution of each algorithm
3. If the existing algorithms solution list is less than the maximum number of population then generate valid schedule arbitrarily.
4. Convert all solutions into the chromosome format
5. Place it as an initial population list

The validity of the initial population is checked using the fitness function.

### 6.3.3 The proposed Fitness function of GSSGA

The objective function of the problem is represented as fitness function in GA. All minimization problems are usually transformed into maximization problems by some appropriate transformation without changing the optimum point. The proposed
new fitness function has three parameters such as the makespan, resource idle time and number of resources, in which the makespan and resource idle time are minimizing functions, while the number of resources is directly proportional to the number of tasks.

The newly developed fitness function $F(i)$:

$$F(i) = \frac{1}{MS} + \frac{NOR}{ARI}$$

where $i$ is the chromosome, MS is makespan, NOR is Number of Resources and ARI is Average Resource Idle time. Using this function the initial population and other scheduling list will be selected. The processing of GSSGA fitness calculations are:

**GSSGA Fitness**

1. From each scheduling solution list get the makespan, average resource idle time and number of resources reserved for scheduling.

2. Apply all these values in the fitness function formula.

3. Place the fitness value in the fitness array $f(i)$, where $i$ is the task, $\forall i \in 1, 2, 3, ..., n$.

The fitness function of each string is to be used to select the best strings using the Roulette wheel method.

**6.3.4 The Selection operation of GSSGA**

The reproduction process forms a new population by selecting strings in the old population based on their fitness values. The selection criterion is that strings with higher fitness value should have a higher chance of surviving to the next generation. Roulette wheel selection method is used to implement reproduction in the GSSGA. This selection method is adopting the methodology which was proposed by
The Roulette wheel selection needs six computations to select the chromosome. Initially the number of population is chosen. The fitness value of each population \((F(i))\) is calculated and arrived at the average fitness function \((\overline{F})\) followed by the computation of the expected count of each string \(A\).

\[ A = \frac{F(i)}{\overline{F}} \]

The probability of each string \(B\) is the expected count \(A\) which is divided by the number of population size \(N\), which is kept fixed in a simple GA.

\[ B = \frac{A}{N} \]

Once the probability is calculated, the cumulative probability \(C\) is computed. The sum of the probability of each string being selected for the mating pool must be one.

\[ C = C + B. \text{ Initially } C = 0. \]

In order to form the mating pool the random number \(D\) between 0 and 1 is generated. The \(E\) refers the selected string, which is based on the \(D\). The \(D\) value ranges available in the cumulative probability \((C)\) strings are selected accordingly. The number of selected strings are counted and placed in \(F\). This way, the string with a higher fitness value will represent a larger range in the \(C\) values and therefore has a higher probability of being copied into mating pool. The following algorithm performs the reproduction operation on a population of strings and generated a new population of strings.

The selection procedures are:

\(GSSGA\) Selection

1. \(\text{For } I = 1 \text{ to } n \text{ do}\)
2. \(\text{Compute } \overline{F}, \text{ followed by calculate } A, B, C, D, E, F\)
3. \(\text{End for}\)
The selected strings are available in E. These strings will be passed to undergo the crossover operation.

6.3.5 The Crossover operation of GSSGA

This operator is to produce a new string from the existing one. The single point crossover is applied to produce a new string in this study. For instance, if the number of strings available is five, then the combinations of crossover among strings are (1,2) (1,3) (1,4) (1,5) (2,3) (2,4) (2,5) (3,4) (3,5) (4,5). Similarly the combination of crossover is considered for any number of populations. From this the best individual string will be selected for the number of population using the fitness function. The probability of crossover rate is 0.65. It is possible to create new strings by exchanging portion of two strings by using the following method.

**GSSGA Crossover**

1. Choose two neighboring parents to crossover
2. Select the crossover position randomly
3. Copy the string from the random position of 2\(^{nd}\) parent to 1\(^{st}\) parent and vice versa.

After performing the crossover operations, all new strings will undergo the validity process. The first step of this validation is checking the existence of total number of tasks. The second step is verifying the precedence constraints. If both conditions are satisfied, those strings are considered as valid strings. The valid strings are selected as the new population for the next generation of the GA. The remaining invalid strings will go to the mutation phase.
6.3.6 The Mutation operation of GSSGA

Mutation can be considered as an occasional random alteration of the value of a string. The invalid strings undergo mutation operation to make as valid strings with one change. The Mutation takes place in any one of the gene, i.e., task number or the resource number of a string. The non-existence of a task in a string will be located in the appropriate place. Similarly if the precedence constraints are not satisfied, swap any one task to fulfill the precedence constraint. Single point mutation is carried out in GSSGA. The probability rate of mutation is 0.01. The following procedure performs the mutation operation, if necessary, on a string and generates new string.

**GSSGA Mutation**

1. *The non-existence task is placed in the correct positions*

2. *The interchanging of any two tasks satisfy the precedence constraint, swaps it.*

By repeatedly executing the *GSSGA Mutation*, new strings are generated.

The GA terminates when it meets the convergence condition.

6.4 Results and Discussion

The proposed Grid Scheduling Strategy using Genetic Algorithm (GSSGA) has been implemented and undergone exhaustive experiments with arbitrary task graphs to analyze its performance.

The results are compared with the well known scheduling algorithms like Min-Min algorithm and Heterogeneous Earliest Finish Time (HEFT) algorithm. Also, the results are found to be better than the EDOS algorithm proposed in the present research study. Among all these algorithms GSSGA is found to be the best in
makespan, resource utilization, communication time and also with higher speedup. The sample schedule is shown in Appendix E. In all the experiments conducted in this research study, the following parameters are assumed.

\[
\begin{align*}
\text{The Population size} & = 20 \\
\text{Crossover Rate} & = 0.65 \\
\text{Mutation Rate} & = 0.01
\end{align*}
\]

### 6.4.1 Makespan

The results of makespan for Min-Min, HEFT, EDOS and GSSGA algorithms with different CCR values are tabulated in Table 6.1, Table 6.2 and Table 6.3. The proposed GSSGA algorithm gives the better result than the other three algorithms. For instance, in the arbitrary task graph with 20 tasks and CCR = 0.2 (Table 6.1) scheduled with 4 resources the GSSGA algorithm completes the schedule with 88 time units but Min-Min algorithm takes 103.1429 time units, HEFT algorithm takes 99.8095 time units and EDOS algorithm takes 93.619 time units. The values in the Table 6.1 are graphically shown in Figure 6.2. Similarly, in the arbitrary task graph with 30 tasks and CCR=0.6 (Table 6.2) scheduled with 4 resources the GSSGA algorithm completes the schedule with 119.333 time units but Min-Min, HEFT and EDOS algorithms take 179.3333, 167 and 147.6667 time units respectively. When the number of tasks are increased to 200 and CCR=1.0 (Table 6.3) scheduled with 4 resources the GSSGA algorithm completes the schedule with 903.999 time units whereas Min-Min, HEFT and EDOS algorithms finish 1000.333, 916 and 910.6661 time units respectively. Hence, the results so obtained is appreciably encouraging in all the strata, namely, in small number of tasks as well as large number tasks.
Table - 6.1 Comparison of GSSGA, Min-Min, HEFT and EDOS algorithms with Makespan (MS), Resource Utilization (RU) and Communication Cost (CC) for arbitrary task graphs – CCR 0.2

<table>
<thead>
<tr>
<th>No of tasks</th>
<th>Algorithm</th>
<th>CCR = 0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Resources</td>
<td>Min-Min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>103.1429</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>96.8095</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>156</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>245.666</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>167.4095</td>
</tr>
<tr>
<td>75</td>
<td>4</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>206.9143</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>557.6667</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>330.238</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>488.0095</td>
</tr>
</tbody>
</table>
Table - 6.2 Comparison of GSSGA, Min-Min, HEFT and EDOS algorithms with Makespan (MS), Resource Utilization (RU) and Communication Cost (CC) for arbitrary task graphs – CCR 0.6

<table>
<thead>
<tr>
<th>No of tasks</th>
<th>Algorithm</th>
<th>CCR = 0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min-Min</td>
<td>HEFT</td>
</tr>
<tr>
<td></td>
<td>Resources</td>
<td>MS</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>131.4285</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>179.3333</td>
</tr>
<tr>
<td></td>
<td></td>
<td>159.6095</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td></td>
<td>218.599</td>
</tr>
<tr>
<td>75</td>
<td>4</td>
<td>392</td>
</tr>
<tr>
<td></td>
<td></td>
<td>251.238</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>549.3333</td>
</tr>
<tr>
<td></td>
<td></td>
<td>308.8952</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>986.3333</td>
</tr>
<tr>
<td></td>
<td></td>
<td>574.0952</td>
</tr>
</tbody>
</table>
Table - 6.3 Comparison of GSSGA, Min-Min, HEFT and EDOS algorithms with Makespan (MS), Resource Utilization (RU) and Communication Cost (CC) for arbitrary task graphs – CCR 1.0

<table>
<thead>
<tr>
<th>No of tasks</th>
<th>Algorithm</th>
<th>CCR = 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Resources</td>
<td>Min-Min</td>
</tr>
<tr>
<td></td>
<td>Resources</td>
<td>MS</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>197.8095</td>
</tr>
<tr>
<td>8</td>
<td>193.1904</td>
<td>10.4991</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>117.333</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>97.8571</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>387.333</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>345.4</td>
</tr>
<tr>
<td>75</td>
<td>4</td>
<td>374</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>305.2857</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>431.6667</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>302.0476</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>1000.333</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>595.6667</td>
</tr>
</tbody>
</table>
Table - 6.4 Comparison of GSSGA, Min-Min, HEFT and EDOS algorithms with Speedup Ratio for arbitrary task graphs

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of Resources</th>
<th>20 tasks</th>
<th>30 tasks</th>
<th>50 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min-Min</td>
<td>HEFT</td>
<td>EDOS</td>
</tr>
<tr>
<td>0.2</td>
<td>4</td>
<td>0.50254</td>
<td>0.51932</td>
<td>0.5537</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.25699</td>
<td>0.25755</td>
<td>0.2704</td>
</tr>
<tr>
<td>0.6</td>
<td>4</td>
<td>0.37536</td>
<td>0.45498</td>
<td>0.4592</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.2024</td>
<td>0.2012</td>
<td>0.2204</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0.2209</td>
<td>0.2329</td>
<td>0.4257</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.1084</td>
<td>0.1155</td>
<td>0.2043</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of Resources</th>
<th>75 tasks</th>
<th>100 tasks</th>
<th>200 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min-Min</td>
<td>HEFT</td>
<td>EDOS</td>
</tr>
<tr>
<td>0.2</td>
<td>4</td>
<td>0.8309</td>
<td>0.8215</td>
<td>0.8354</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.5908</td>
<td>0.6123</td>
<td>0.6324</td>
</tr>
<tr>
<td>0.6</td>
<td>4</td>
<td>0.7458</td>
<td>0.7542</td>
<td>0.7643</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.5334</td>
<td>0.5461</td>
<td>0.5541</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0.7185</td>
<td>0.7444</td>
<td>0.7612</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.4034</td>
<td>0.4583</td>
<td>0.4843</td>
</tr>
</tbody>
</table>
Figure 6.2 Makespan of GSSGA, Min-Min, HEFT and EDOS algorithms for arbitrary task graphs
6.4.2 Resource Utilization

The results of resource utilization for Min-Min, HEFT, EDOS and GSSGA algorithms with different CCR values are tabulated in Table 6.1, Table 6.2 and Table 6.3. The proposed GSSGA algorithm gives the better results than the other algorithms. For instance, in the arbitrary graph with 20 tasks and CCR = 0.6 (Table 6.2) scheduled with 4 resources the GSSGA algorithm utilizes the resources with 47.6261% but Min-Min algorithm utilizes the resources 37.8532%. HEFT algorithm utilizes 45.9815% and EDOS algorithm utilizes 46.5642%. The values in the Table 6.2 are graphically shown in Figure 6.3. Similarly, in the arbitrary task graph with 30 tasks and CCR=0.6 scheduled with 4 resources the GSSGA algorithm utilizes the resources with 70.041% but Min-Min, HEFT and EDOS algorithms utilize 46.5613%, 51.8463% and 56.15124% respectively. When the number of tasks are increased to 200 and CCR=0.6 scheduled with 4 resources the GSSGA algorithm utilizes the resource 90.3644% whereas Min-Min, HEFT and EDOS algorithms utilize the resources 87.66475%, 88.3644% and 88.512% respectively. Hence, GSSGA algorithm yields comparatively better results even for larger number of tasks as well as for smaller tasks.
Figure 6.3 Resource Utilization of GSSGA, Min-Min, HEFT and EDOS algorithms for arbitrary task graphs
6.4.3 Communication Time

Table 6.1, Table 6.2 and Table 6.3 summarize the results of communication time for Min-Min, HEFT, EDOS and GSSGA algorithms with different CCR values. The proposed GSSGA algorithm consistently gives better results than other algorithms. For instance, in the arbitrary task graph with 20 tasks and CCR = 1.0 (Table 6.3) scheduled with 4 resources the GSSGA algorithm communication time is 91 units but Min-Min, HEFT and EDOS algorithms communication time are 215, 167 and 95 units respectively. The values in the Table 6.3 are graphically shown in Figure 6.4. Similarly, in the arbitrary task graph with 30 tasks and CCR=1.0 for the GSSGA, the communication time is 175 time units but Min-Min, HEFT and EDOS algorithms take the communication time as 209, 187 and 212 units respectively. When the number of tasks are increased to 200 and CCR=1.0 scheduled with 4 resources the GSSGA algorithm communication time is 3214 units whereas for Min-Min, HEFT and EDOS algorithms communication time are 3344, 3675 and 3691 units respectively. Hence, GSSGA algorithm yields comparatively better results for all ranges of task graphs.
Figure 6.4 Communication cost of GSSGA, Min-Min, HEFT and EDOS algorithms for arbitrary task graphs
6.4.4 Speedup Ratio

The speedup of the proposed GSSGA algorithm has been analyzed with Min-Min, HEFT and EDOS algorithms with different CCR values. The speedup values of GSSGA are tabulated in Table 6.4 and the arbitrary task graphs with CCR-0.2 are graphically represented in Figure 6.5.

![Graphs showing speedup ratio for different task graphs with CCR 0.2](image1.png)

Figure 6.5 Speedup Ratio of GSSGA, Min-Min, HEFT and EDOS algorithms for arbitrary task graphs

155
Figure 6.5 shows the comparison of the proposed GSSGA algorithm with Min-Min, HEFT and EDOS algorithms with CCR = 0.2. In all cases, it is found that the performance in GSSGA algorithm is better than the other algorithms.

The speedup of GSSGA with varying number of tasks as 20, 30, 50, 75, 100 and 200 with different CCR values is also studied and the variation of speedup is shown in Figure 6.6.
From the Figure 6.6, it is observed that the speedup increases proportionally with the number of tasks. Hence, it may be concluded that computational intensive application (CCR=0.2) can be completed earlier than the communication intensive application (CCR=1.0).

6.5 The Scheduling Tool

The proposed algorithms such as LCTSA, EDOS, TDB-EMOS and GSSGA as well as the popular algorithms in the literature, namely, Min-Min, HEFT and EDS-G are implemented in the simulated grid environment. This tool is used to generate DAG with different number of tasks by different values of communication and computation time arbitrarily. The Min-Min, HEFT, LCTSA, EDOS, TDB-EMOS and GSSGA are implemented and tested with different number of tasks with varying number of CCR values. This tool is used to store the DAG and the resource sets with its computational speed. The functional architecture of the scheduling tool is shown in Figure 6.7.

Figure 6.7 Scheduling Tool
The architecture consists of input module, scheduling model and Display model. The scheduling algorithm developed during the course of research study are implemented and tested with DAG model / workflow model task graphs.

The input model feeds the arbitrary DAG generated by DAGEN with a required number of tasks and a specified CCR value or a regular task graph. This model also specifies the number of resources available (which are reserved in advance) in the grid environment. The generated DAG is stored in a file for conducting other experiments.

The scheduling model shows a menu in Graphical User Interface. Showing the names of all the scheduling algorithms, viz., Min-Min, HEFT, LCTSA, EDOS, EDS-G, TDB-EMOS and GSSGA. When the user selects any one of the algorithms, the generated DAG/Regular task graph will be scheduled, Then the Display module gives the list of optimal schedule with tasks and resources. Also it shows the performance metrics like makespan, resource utilization, communication cost and speedup. While scheduling the task graph using GSSGA the converged optimal schedule with number of iteration will be displayed. The sample screen shots of the scheduling tool are shown in Appendix F.

6.6 Conclusion

In this chapter a new Grid Scheduling Strategy using Genetic Algorithm (GSSGA) has been proposed to reduce optimal schedule with the available resources. Genetic Algorithm is an evolutionary approach to derive the optimal solution from the existing schedules. Hence, the initial population consist of chromosomes [schedules]
derived from various list scheduling algorithms. The genetic operators are applied suitably in various iterations and finally the optimal schedule is converged.

In this methodology a new fitness function has been developed to validate the strings to be taken for the next iteration. After conducting experiments with different types of task graphs the GSSGA gives consistent performance with lower makespan and higher resource utilization.

A scheduling tool has been developed in a Graphical User Interface by combining all the algorithms studied in the present research work. This will be helpful to the users to generate optimal schedule either by using list algorithm or Genetic Algorithm in a more user friendly manner.