CHAPTER 3

DISK SCHEDULING

3.1 INTRODUCTION

FCFS algorithm arranges the read/write requests in the order of the requests. No reordering of work queue is done in FCFS. There is no starvation as every request is serviced. But the performance of this algorithm is poor.

In SSTF, after a request has been serviced the R/W head goes to the closest request in the work queue, regardless of direction. It reduces the total seek time compared to FCFS. The disadvantage of SSTF is starvation, that is, the R/W head stays in one area of the disk if very busy.

In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the center, servicing requests as it reaches each cylinder, until it gets to the center. From this end, the direction of head movement is reversed and servicing continues. The head continuously scans back and forth across the disk. C-SCAN moves the head from one end of the disk to the other, servicing the requests along the way. When the head reaches the other end, however it immediately returns to the beginning of the disk without servicing any requests on return trip.

In LOOK, the arm goes only as far as the final request in each direction. Then it reverses immediately, without going all the way to the center. In C-LOOK, arm moves only as far as final request and then the arm goes back to the starting of the disk without servicing any request in return trip.
With an example, all the algorithms can be compared. Let the requests in the work Queue be 23, 89, 132, 42 and 189. Let there be 200 cylinders with numbers from 0 to 199 in the system. Let the disk head start at number 100. Total length travelled by disk arm for the algorithms are illustrated in Figures 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6.
Figure 3.3 SCAN

Figure 3.4 C-SCAN
3.2  PROBLEM FORMULATION

Consider a set of $n$ disk requests $r = \{r_1, r_2, \ldots, r_n\}$. Finding a feasible schedule $r': rw(1) \ rw(2) \ rw(y) \ rw(n)$ with minimal completion time and minimal number of requests missed, is the goal of real-time disk schedulers.
The index function \( w(i) \), for \( i = 1 \) to \( n \), is a permutation of \( \{1, 2, \ldots, n\} \).

For any disk request, time is required to seek the track where the request is located. This time is called as seektime. It is calculated as given in Equation (3.1).

\[
\text{seektime} = \text{abs}(\text{cheadpos} - \text{tracknum})
\]  

where

- \text{cheadpos} - Current head position
- \text{tracknum} - Track number of the request

To transfer the request from disk to buffer, some time is spent which is called as transfer time. The completion time of a request is the time to complete the request. It is found using Equation (3.2)

\[
\text{c\_time} = \text{cur\_time} + \text{seektime} + \text{transtime}
\]

where

- \text{c\_time} - Completion time of the request
- \text{cur\_time} - Current time
- \text{seektime} - Seek time
- \text{transtime} - Transfer time, time taken to transfer the data

Deadline time is the latest time at which disk request should be completed.

Throughput of the disk scheduling is calculated using Equation (3.3)

\[
\text{Throughput} = \frac{b}{\text{c\_time}}
\]

where

- \( b \) - Data size of the request

Since throughput depends on the completion time of the disk requests, our objective is to minimize the completion time. For real-time applications, missed disk requests are to be reduced.

### 3.3 PROPOSED SOLUTION STRATEGIES

Disk scheduling is formulated with the objectives of minimizing the completion time and minimizing the number of missed tasks. The
approach MOGA is customized by including elitism. For HGA and Micro-GA, the crossover operator is modified.

3.3.1 MOGA

MOGA is given in Figure 3.7 and described in this sub section.

1. Initialize the population.
2. Traverse the ordering from left to right calculate the fitness values
3. Select the individuals by applying elitism.
4. Perform cross over and mutation.
5. Perform selection on ordered individuals.
6. Repeat steps 2-5 till the stopping criterion is reached.

**Figure 3.7 Steps for MOGA**

**Population Initialization**

The population is generated randomly with the given chromosomes. The chromosomes are the given read/write requests. The population size is set to 200 to provide all combination of requests arrangement in the sequence. Figure 3.8 shows the algorithm for population initialization.

**Input:** Number of read/write requests (n),
track number of each request.

**Output:** Population (Initialized population)

```
popsize= 200;
for i: 0 to popsize - 1
  for j: 0 to n-1
    Population[i] [j]=rand(n);
Return Population;
```

**Figure 3.8 Algorithm for population initialization in MOGA**
Coding Scheme

The coding scheme is used to correct the repetitions in sequence of requests. Each chromosome shows a sequence of numbers between 0 and n-1 and each genome points to the index of an array which includes unscheduled tasks. Figure 3.9 shows how the coding scheme is implemented.

**Input:** The sequence of chromosomes(C[0-(n-1)])

**Output:** Sequence S

S: emptysequence()

Q:{0, 1,2,3, … n-1}

for i:0 to n-1/

T=Mod(C[i], length(Q));

S=S+Q[T];

Eliminate Tth element of Q;

/

Return S;

Figure 3.9 Algorithm for coding scheme in MOGA

To correct the repetition in the sequence 3 2 2 0 1, the coding scheme is applied. The results of applying Algorithm2 on the sequence 0, 1, 2, 3, 4 are shown in the Table 3.1.

Table 3.1 Example of coding scheme in MOGA

<table>
<thead>
<tr>
<th>I</th>
<th>Q</th>
<th>C[i]</th>
<th>Length[Q]</th>
<th>T</th>
<th>Q[T]</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0,1,2,3,4}</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>&lt;2&gt;</td>
</tr>
<tr>
<td>1</td>
<td>{0,1,3,4}</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>&lt;2,0&gt;</td>
</tr>
<tr>
<td>2</td>
<td>{1,3,4}</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>&lt;2,0,1&gt;</td>
</tr>
<tr>
<td>3</td>
<td>{3,4}</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>&lt;2,0,1,4&gt;</td>
</tr>
<tr>
<td>4</td>
<td>{3}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>&lt;3,0,2,1,4&gt;</td>
</tr>
</tbody>
</table>


**Fitness Function**

The proposed fitness function is presented in this section. The completion time is introduced in the fitness evaluation procedure. The completion time is the total time that the last request execution order completes its work. As an instance, the completion time for the order in the sequence 0, 1, 2, 3, 4 is 16 because the last request (R4) completes its work in time 16 and the maximum deadline of the sequence is 18ms.

The goal is to find an order for executing some number of requests such that the completion time and the number of missed tasks to be minimized. In this way, a penalty function has been designed which guarantees the best completion time with regard to the number of missed tasks. The equation 3.4 shows the formula that can be used to evaluate the fitness of each arrangement.

\[
\min f(c_{\text{time}}, m_{\text{deadline}}) = \sum_{i=0}^{n-1} (c_{\text{time}})_i + m_{\text{deadline}} \times \text{miss}
\]

where  
- \( n \) - Number of requests  
- \( m_{\text{deadline}} \) - deadline for each request  
- \( \text{miss} \) - Number of missed tasks

The equation 3.4 shows the formula that can be used to evaluate the fitness of each arrangement.

Fitness function is minimized when miss is zero. The main factor which distinguishes the feasible order is penalty value. When the miss increases the fitness gets increased. By using this fitness function, when missed tasks are minimized, number of completed tasks is maximized. By using this fitness function, when missed tasks are minimized, number of completed tasks is maximized. On the other hand, by minimizing the completion time, the throughput increases. Therefore, the proposed fitness
function increases throughput, decreases miss tasks, and decreases completion time simultaneously.

**Sorting**

After calculating the fitness function for the generated population, the genomes are sorted in the population according to their fitness value. The genomes with least fitness value are the optimal arrangement of read/write requests.

**Selection**

The main idea of selection is to make better individuals get higher chance. This is a way of choosing members from the population of chromosomes in a way that is proportional to their fitness. Elitism selection method is used in our proposal. In elitism, esize is assigned randomly and those numbers of genomes are taken directly to next generation and no genetic operators applied to it. Figure 3.10 implements selection method, elitism.

| Input: The sorted population (population). |
| Output: Selected population for next generation (Population). |

esize=10;

\[
\text{for } i : 0 \text{ to } esize - 1 \\
\quad \text{Population1}[i].\text{fitness}=\text{population}[i].\text{fitness}; \\
\quad \text{for } j : 0 \text{ to } n-1 \\
\quad \quad \text{Population1}[i].\text{chromosome}[j]=\text{population}[i].\text{chromosome}[j]; \\
\]

return Population

**Figure 3.10 Algorithm for applying Elitism in MOGA**

**Crossover**

Crossover is a genetic operator that combines two chromosomes to produce a new chromosome (offspring). The idea behind crossover is that the
new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. One-point crossover, Two-point crossover, Cut and splice, Uniform Crossover and Half Uniform Crossover and Three parent crossover are the various crossover operators used. For some applications, a direct swap may not be possible. For example, in this work, real coded GA is used. (i.e. the chromosomes are represented using real values). After crossover is applied, there is a possibility of missing some requests and some requests may be repeatedly executed. To avoid this, Crossovers such as partially matched crossover (PMX), cycle crossover (CX) position-based crossover may be used.

For this work, CX is used. CX performs recombination under the constraint that each allele should come from one parent or the other. Here instead of choosing crossover sites, we start from at the left until we reach the individual from which the operation started. This completes a cycle. Remaining individuals are filled from the other parent.

1. Make a cycle of alleles from P1 in the following way.
   a. Start with the first allele of P1.
   b. Look at the allele at the same position in P2.
   c. Go to the position with the same allele in P1.
   d. Add this allele to the cycle P1’ and also add the allele in same position from P2 to P2’
   e. Repeat step b through d until you arrive at the first allele of P1.
2. Remaining alleles in P1’ and P2’ are filled from P2 and P1 respectively.

|Figure 3.11 Procedure for CX in MOGA|

Let P1 and P2 be two parents coded as given below.

P1 9 8 2 1 7 4 5 10 6 3
P2 1 2 3 4 5 6 7 8 9 10
Cycle of alleles is made as given below.

P1' 9 _ _ 1 _ 4 _ _ 6 _
P2' 1 _ _ 4 _ 6 _ _ 9 _

Remaining alleles in P1’ and P2’ are filled from P2 and P1 respectively.

P1' 9 2 3 1 7 4 8 9 6 10
P2' 1 8 2 4 7 6 5 10 9 3

In modified CX, remaining alleles in P1’ and P2’ are filled with the missing individuals in the order. This approach avoids repetition of alleles in individuals which is needed for real coded GA and it can give different combination of individuals instead of retaining from the parent. Modified CX is illustrated in Figure 3.12.

Each allele comes from one parent together with its position.

Procedure:
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the individual at the same position in P2.
   (c) Go to the position with the same individual in P1.
   (d) Add this individual to the cycle P1’ and also add the individual in same position from P2 to P2’
   (e) Repeat step b through d until you arrive at the first individual of P1.
2. Remaining individuals in P1’ and P2’ are filled with the missing individuals.

**Figure 3.12 Procedure for modified CX in MOGA**

As an example, let P1 and P2 are two parents as given below.

P1 9 8 2 1 7 4 5 10 6 3
P2 1 2 3 4 5 6 7 8 9 10
Retaining alleles from P1 and P2 starting from left until first allele is reached

\[ \begin{align*}
P1' & \quad 9 \quad _{\_} \quad 1 \quad _{\_} \quad 4 \quad _{\_} \quad 6 \quad _{\_} \\
P2' & \quad 1 \quad _{\_} \quad 4 \quad _{\_} \quad 6 \quad _{\_} \quad 9 \quad _{\_}
\end{align*} \]

Remaining alleles are filled with the missing alleles in order.

\[ \begin{align*}
P1' & \quad 9 \quad 2 \quad 3 \quad 1 \quad 5 \quad 4 \quad 7 \quad 8 \quad 6 \quad 10 \\
P2' & \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10
\end{align*} \]

**Mutation**

Mutation is a genetic operator that alters one or more chromosome values in a genome from its initial state. This can result in entirely new genome. With these new chromosome values, GA may arrive at better solution. Mutation is an important part of the genetic search as it helps to prevent the population from stagnating at any local optima. Swap mutation is used in this proposal because it suits well for the real time applications and prevents local optima in an efficient way. The procedure is given as follows.

- Pick two chromosomes at random and swap their positions.
- Preserve most of adjacency information

For the parent 1 2 3 4 5 6 7 8 9, the resulting offspring would be 1 5 3 4 2 6 7 8 9. The offspring is illustrated in Figure 3.13.

![Figure 3.13 Swap mutation](image)

**Stopping Criteria**

For minimum number of requests say 5, convergence occurs between 10th and 15th generation. For the requests more than 10, convergence is obtained between 25th and 30th generation. Hence, stopping criteria is set as 30 numbers of generations.
3.3.2 HGA

To develop HGA, the features of SA are combined to the basic MOGA framework. The capability of SA for selecting the fittest candidate solutions are used as input to the cross-over module. Though SA takes some time to cool down to the equilibrium state, it eliminates the dependency of the selection process on a complete pool of candidate solutions required in conventional method at the selection stage. Both SA and MOGA are randomized guided search methods which when combined result in HGA.

In the new method of selection, called stochastic selection a chromosome with a value $x_i$ is considered from a pool $P(g)$ of generation $g$ and is selected based on Boltzmann probability distribution function. Let it be assumed that $f_{max}$ is the fitness of the currently available best chromosome. If the next chromosome has fitness $f(x)$ such that it is greater than $f_{max}$, then the new chromosome is selected otherwise it is selected with Boltzmann probability (Gen and Lin 2005 a).

$$P(\exp \left[\frac{-(f(y)-f(x))}{T}\right] \geq \text{random}\ [0, 1])$$

where

$$T = T_0 \ (1- \alpha) \ K \text{ and } K = (1 + (g / G) \ *100)$$

The value of $\alpha$ can be chosen from the range $[0, 1]$, and $T_0$ from the range $[5, 100]$. The Equation (3.6) shows that the value of $T$ decreases exponentially or at logarithmic rate with increase in the value of $g$ and hence the value of the probability $P$. This is significant in terms of convergence. The final state is reached when computation approaches zero value of $T$, i.e. the global solution is achieved at this point. In the proposed HGA algorithm, the probability that the best string is selected and automatically included as a member of the selected population is very high.
Begin
  g = 0
  initialize (T, P (g))
evaluate P (g) using fitness function
fmax maximum fitness of P (g)
termination_condition = false
while (NOT termination_condition) do
  begin
    g = g + 1
    for i = 1 to N do{
      if fmax-f(xj)<=0
        then select xj from P (g) and set fmax to f (xj)
      else if (exp [-f(y)-f(x) / T ] >= random [0, 1])
        then select xj from P (g)
      else select x corresponding to fmax
    }
crossover
mutation
evaluate P (g + 1) using fitness function
lower T
end
end

Figure 3.14 Proposed hybrid algorithm

However, elitism is suggested to eliminate the chance of any undesired loss of information during the mutation stage. The proposed hybrid algorithm is shown in Figure 3.14.

3.3.3 Micro-GA

To fine tune the speed of MOGA, micro-GA is used. In micro-GA the population size is maintained as small as possible. The structure of micro-GA is illustrated in Figure 3.15. This sub section also explains micro-GA methodology.
The first step in micro-GA is population initialization. The code for population initialization is same as that of MOGA described in Section 3.3.1, Figure 3.7. Coding scheme is used to decode the randomly initialized population to the sequence which represents the order of the tasks. Same coding scheme as that of MOGA given in Section 3.3.1, Figure 3.8 is used for micro-GA coding scheme.
Micro-GA Cycle

At the micro-GA cycle level, the N individuals of initial micro-population are obtained randomly from either replaceable or non-replaceable population. Based on the fitness value the rank is calculated for each individual. After selection, the cycle crossover and swap mutation are applied. The cycle is continued until convergence is reached.

Fitness Calculation

The aim of disk scheduling is to reduce the completion time and the number of missed tasks. The fitness function is designed as given in Section 3.3.1.

Table 3.2 A sample disk scheduling problem

<table>
<thead>
<tr>
<th>Task name</th>
<th>Cylinder</th>
<th>Deadline(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>T1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>T2</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>T4</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

The decoded sequence obtained from Table 3.2 is \{3, 0, 2, 4, and 1\}

Table 3.3 Task parameters to calculate fitness function

<table>
<thead>
<tr>
<th>Task</th>
<th>Current head position</th>
<th>Current time (mS)</th>
<th>Seek time (mS)</th>
<th>Data transfer time (mS)</th>
<th>Completion time (mS)</th>
<th>If task Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>3</td>
<td>0+1+3=4</td>
</tr>
<tr>
<td>T0</td>
<td>1</td>
<td>4</td>
<td>5-1=4</td>
<td>3</td>
<td>4+4+3=11</td>
<td>No</td>
</tr>
<tr>
<td>T2</td>
<td>5</td>
<td>11</td>
<td>6-5=1</td>
<td>3</td>
<td>11+1+3=15</td>
<td>Yes</td>
</tr>
<tr>
<td>T4</td>
<td>5</td>
<td>11</td>
<td>3-5=2</td>
<td>3</td>
<td>11+2+3=16</td>
<td>No</td>
</tr>
<tr>
<td>T1</td>
<td>3</td>
<td>16</td>
<td>3-3=0</td>
<td>3</td>
<td>16+0+3=19</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Since task 3 has the cylinder number 1, the schedule starts from task 3. Let the current head position and current time be 0. Task 3 needs one
disk seek (from Table 3.2) and 3ms to transfer the data. Completion time of the task is the sum of seek time, current time and data transfer time. Hence the task is completed in 4 ms which is less than its deadline. So task 3 is not missed. Now current head is 1 and current time is 4. For task 2, current head is 1 and current time is 4. The completion time for T2 is greater than the deadline (i.e. 15 > 12). Hence, task T2 is missed. If a task is missed it is not considered for current head position and current time. This procedure is repeated for all the tasks in the sequence and the final completion time obtained is 16 ms. Number of missed tasks is 2 (from Table 3.3). The maximum deadline given in Table 3.2 is 18 ms.

\[ \text{Fitness} = 16 + 2 \times 18 = 52 \]

**Selection of micro-Population**

In order to reduce the complexity of calculations micro-pop is selected to which genetic operators are to be applied. The individuals are selected randomly from the replaceable and non-replaceable portions of the population. The best individual is retained by selection process. It is given by Figure 3.16.

The selection method used in proposed approach is Elitism. The best individual belonging to the micro-population is passed unaltered to the next micro-generation. This is the first form of elitism. In the second form of elitism the representative solution of the micro-GA cycle replaces a randomly selected individual of the replaceable portion. In the third form of elitism non dominated solutions are used to update the replaceable portion. The crossover and mutation operator are implemented as given in Section 3.3.1.
**Input**: Order $T$

**Output**: micro_pop $pop1$

Elitism

$k = \text{random()} \mod \text{popsize}$

$pop1[k].fitness = T[k].fitness$

for $j: 0$ to $num$

$pop1[k].chrom[j] = T[k].chrom[j]$

return $pop1$;

---

**Figure 3.16 Micro-pop selection**

---

**Genetic Operators**

The operations crossover, mutation and selection are implemented as given in Section 3.3.1.

**External Memory**

The external memory contains the best solution from each micro-GA cycle. The nominal solution of the micro-GA cycle is copied in the external memory. However, this happens only when the nominal solution is non-dominated with respect to the individuals already belonging to the external memory.

**Convergence**

In proposed approach, the micro-GA cycle stops when convergence is reached in term of number of iterations. Once the maximum number of iterations is reached, a representative individual is selected as a nominal solution. From experimental observation convergence is achieved within in 30 generations. So the number of iterations fixed is 30.
3.4 IMPLEMENTATION AND RESULTS

All implementations are performed on a personal computer with 2.26GHz of CPU and 3GB of RAM in C environment. 200,000 disk request sets are generated randomly, each consisting of various sequences of requests. The initial population size is fixed as 500. Initially, elitism rate is selected as 10%. The test has been conducted for 5, 10 and 15 disk requests. Several runs are performed.

The deadlines of requests are calculated as $\text{TimeMultiplier} \times \text{TimeoutBase} + \text{Timeout} \times (\text{Size}/36\text{KB})$ where TimeMultiplier is a uniform random number in interval [1-5] and the value of TimeoutBase is 550 for each task. Also, the value of Timeout is 10 and the size parameter is the size of requested data by each request.

To make the work as real-time, files are taken as the input. Different types of files such as text (.txt), images (.jpg, .bmp), media (.wma, .avi) are used. The size of the file is derived using ‘ftell’ function and it is used for the calculation of deadline and throughput. Table 3.4 shows the parameters and the values.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of chromosomes</td>
<td>Number of read/write requests</td>
</tr>
<tr>
<td>Population size</td>
<td>200</td>
</tr>
<tr>
<td>Elitism Rate</td>
<td>10%</td>
</tr>
<tr>
<td>Crossover type</td>
<td>Modified Cycle crossover</td>
</tr>
<tr>
<td>Mutation</td>
<td>Swap mutation</td>
</tr>
<tr>
<td>Generations</td>
<td>30</td>
</tr>
<tr>
<td>Micro-Population size (For micro-GA)</td>
<td>7</td>
</tr>
</tbody>
</table>
Parameter Setting

Effect of population size on fitness

The parameter, population size is fixed as 200 based on the experimental observation. The minimized fitness value is obtained when the population size is maximized. The experiment is done for different set of population sizes (10, 50, 100, 150, 200, 250 and 350) with different set of requests (5, 10 and 15). The average fitness is taken from the observation. The minimized fitness is obtained in population size of 200. There is no change in fitness with larger population size.

Figure 3.17 shows the effect of population size on fitness value. It is clear from the graph that fitness is minimized when population size is maximized.

Effect of micro population size on time taken for calculation

The micro-pop size is fixed as 7 in experiment based on the observation that time taken for calculation increases when the micro-pop size gets increased. The experiment is performed with different micro-pop sizes with different number of requests. When micro-pop size gets decreased,
recombination between the individuals cannot be produced. So, on an average micro-pop size is fixed as 7. Figure 3.18 shows the effect of micro-pop size on the time taken for calculation.

![Figure 3.18 Micro population size vs. Time taken for calculation](image)

**Effect of Elitism rate**

Figures 3.19 shows the effect of including elitism in MOGA and micro-GA respectively. For MOGA, an example of 5 requests is shown. Elitism is done to retain the best individual in the next generation. If elitism rate is 0, then the fitness converges only after many generations or it never converges. If the elitism rate is fixed, then fitness value converges quickly. However, if elitism rate is increased further, say, 2, 3 or 4, there is no improvement.

In micro-GA, if the elitism rate is zero, the convergence is obtained after many generations. As an example, 10 requests are considered for comparing the elitism rates. If elitism size is fixed as 10, the fitness value converges quickly. If the value is increased to 15, 20, etc., there is no further improvement in convergence. Throughout the experiment, esize is fixed as 10 for micro-GA. The result is illustrated in Figure 3.20.
<table>
<thead>
<tr>
<th>Scheduling sequence</th>
<th>Fitness value</th>
<th>Scheduling sequence</th>
<th>Fitness value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 9 1 7 6 5 3 2 8</td>
<td>345</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>9 1 7 6 5 3 2 8 0 4</td>
<td>267</td>
</tr>
<tr>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
</tr>
<tr>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
<tr>
<td>3 2 8 0 4 6 5 9 1 7</td>
<td>246</td>
<td>0 4 9 7 1 6 5 3 2 8</td>
<td>311</td>
</tr>
<tr>
<td>0 4 9 3 2 8 7 1 6 5</td>
<td>367</td>
<td>6 5 3 2 8 0 4 9 1 7</td>
<td>209</td>
</tr>
</tbody>
</table>

Figure 3.19 Comparison of Elitism rate in MOGA
<table>
<thead>
<tr>
<th>Scheduling sequence</th>
<th>Fitness value</th>
<th>Scheduling sequence</th>
<th>Fitness value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 2 0 3 1</td>
<td>223</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 4 2 1 3</td>
<td>374</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 4 0 1 3</td>
<td>284</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 2 0 1 3</td>
<td>284</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 1 0 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 1 0 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 2 1 3 0</td>
<td>308</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 3 4 2 1</td>
<td>361</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 4 0 1</td>
<td>393</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 4 1 0 3</td>
<td>332</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>3 1 2 4 0</td>
<td>398</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 2 4 3 1</td>
<td>313</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 3 4 2 1</td>
<td>361</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 4 2 1 3</td>
<td>374</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>3 4 2 0 1</td>
<td>361</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 0 1 3 2</td>
<td>369</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 3 1 2 0</td>
<td>430</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 1 4 0</td>
<td>430</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 2 1 3 0</td>
<td>308</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 4 3 0 1</td>
<td>271</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 1 0 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>0 2 4 3 1</td>
<td>313</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>3 4 2 1 0</td>
<td>446</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 4 0 1 3</td>
<td>284</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 0 1 3 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 0 1 3 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>4 2 1 3 0</td>
<td>308</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 0 1 3 4</td>
<td>346</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 0 4 3 1</td>
<td>345</td>
</tr>
<tr>
<td>2 4 0 3 1</td>
<td>223</td>
<td>2 3 1 0 4</td>
<td>346</td>
</tr>
</tbody>
</table>

Figure 3.20 Effect of esize in micro-GA
Pareto Front produced by MOGA and Micro-GA

For any multi-objective problem, the pareto front which provides the set of solutions is very crucial (Tan and Khor 2003). Figure 3.21 provides the pareto front for the GA based approaches. It shows number of missed tasks vs. completion time which are the conflicting objectives. Pareto front finds a trade-off between these two objectives.

![Figure 3.21 Pareto front produced](image)

Comparison of GA based approaches and Traditional algorithms

The experiment showed a significant improvement in completion time and throughput in comparison with FCFS, SCAN, C-SCAN, LOOK, C-LOOK and SSTF scheduling algorithms.

Figure 3.22 shows the comparison of completion time between the GA based approaches and other traditional algorithms with different sets of requests (5, 10 and 15). FCFS takes the highest completion time for the schedules. The performance of GA based approaches is almost equal to that of SSTF. For all the requests, 5, 10 and 15, the completion time for SSTF is equal to that of GA based approaches. However, SSTF does not handle the deadlines and consequently, starvation is very high. The proposed approaches...
not only optimize the completion time but also the number of missed tasks. This leads to the optimization of throughput and starvation.

![Figure 3.22 Traditional Algorithms vs. GA based Approaches](image)

**Figure 3.22 Traditional Algorithms vs. GA based Approaches**

It is well known that FCFS, SSTF, LOOK and C-LOOK may lead to starvation while servicing the requests. SCAN has comparatively poor performance regarding response time. C-SCAN is able to provide uniform wait time for all head positions. However, when the head reaches one end, it returns to the beginning without serving any request. It leads to starvation. Since the genetic algorithm based approaches need lower completion time than other traditional algorithms, all the requests are completed. Hence starvation is avoided. It is found that genetic algorithm based approaches not only able to minimize the completion time but also minimize the number of missed tasks.

### 3.5 SUMMARY

Disk scheduling algorithm is optimized by using GA based approaches. From the numerical results, it is found that GA based approaches require less completion time than other traditional algorithms. It is shown that
GA based approaches outperform traditional disk scheduling algorithms by optimizing multiple objectives simultaneously. Moreover, the objectives are conflicting. Micro-GA outperforms the other two approaches. It is able to produce the minimized completion time and minimum number of missed tasks in the schedule.