5.0 INTRODUCTION

In this Chapter, design and development of a multi-objective genetic algorithm is presented. Key issues such as chromosome encoding, selection process, selection of appropriate genetic operators and preservation of non-dominated solutions found during the search are addressed. In addition to addressing the above issues, a local search mechanism is proposed to improve the performance of GA, a new non-domination ranking procedure and a new method to preserve diversity among the non-dominated front members are presented.

Genetic algorithm is a general purpose search optimization algorithm suitable for multi-objective optimization due to its parallel processing approach and its structure by which it is able to return a set of multiple Pareto optimal solutions. The objective is to search non-dominated solutions iteratively by a parallel process using a set of feasible solutions called population. The population undergoes evolution based on the survival process. Some of the important terminologies used in GAs are presented below.

A chromosome is a candidate solution or a schedule represented by a sequence of binary digits or integers or real numbers or dispatching rules depending upon the type of application. A chromosome consists of genes, each of which represents a unique feature of the schedule. A collection of chromosomes is called a population. The number of chromosomes, known as population size, is an important parameter of GA. Fitness is the performance measure used to evaluate chromosomes in the population. The definition of the fitness function is crucial because the desirability of the feature is described by fitness value of the chromosome. Higher fitness value indicates better performance of the individual. Hence, individuals with higher fitness will have better chances of survival.

The three fundamental GA operators are reproduction, crossover and mutation. The key aspects in the design of GAs are representation scheme, fitness assignment procedure and selection of crossover and mutation operators. Initially, a population of chromosomes is generated and their fitness functions are evaluated. The population undergoes iterative
evolution process called generation. At the beginning of every generation, an intermediate population is created by copying each chromosome from the current population zero, one or more times according to its fitness. This process is called selection. Commonly used selection methods in GA are the fitness proportionate selection (Goldberg, 1989) and the tournament selection (Goldberg and Deb, 1991). In every generation, genetic operators such as crossover and mutation generate new solutions. Crossover operator combines two parents to produce one or more offspring. Mutation operator changes chromosomes to produce a slightly modified solution. The entire set of generations is called a run.

Step 1: Randomly generate the initial population of size PS
Step 2: Evaluate each chromosome in the population to determine fitness of individuals
Step 3: Repeat the following steps until PS number of offspring have been generated
  3.1 Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness
  3.2 Crossover the pair with probability $P_x$ to yield two offspring
  4.3 Mutate the two offspring with probability $P_m$ and place the resulting chromosomes in the new population
Step 4 Replace the current population with the new population
Step 5: If termination criterion is not met, go to step 2, else output population members and their fitness.

Figure 5.1 Simple genetic algorithm

During the last 10 years, a number of different GA approaches were proposed to solve multi-objective optimization problems. Most of the applications are meant for solving continuous test functions. Scheduling problems are discrete optimization problems with complex technological constraints, which require selection or design of useful operators to solve multi-objective scheduling problems better. In the following sections, the representation scheme, selection method and genetic operators considered for study in the proposed research are described.
5.1 REPRESENTATION SCHEMES FOR JSSP

The representation scheme is the most important and difficult aspect of GAs when they are used for solving JSSPs. This is due to the technological constraints of the JSSP. Chromosome encoding or representation determines how a schedule can be represented. A chromosome is a combination of symbols and characters called genes. The representation defines genetic operators, which would have the required characteristic of passing some characteristic structure from parent to offspring. Nine representation schemes have been proposed in the literature (Cheng et al., 1996) for the JSSPs. These methods are classified with regard to their structure as follows:

5.1.1 Direct Representation Schemes

1. Operation-based representation
2. Job-based representation
3. Job pair relation based representation
4. Completion time-based representation
5. Random keys representation

5.1.2 Indirect Representation Schemes

1. Priority rule-based representation
2. Preference-list based representation
3. Disjunctive graph-based representation
4. Machine-based representation

In direct approach, a schedule is encoded directly into a chromosome and GAs are used to evolve those chromosomes to find out a better schedule. Direct representation approaches such as operation based (Fang et al., 1993, Gen et al., 1994; Kubota, 1995, Bierwirth, 1995; Wang and Zheng, 2001), job based (Holsapple et al., 1993), job pair relation based (Paredis, 1992), completion time based (Yamada and Nakona, 1992) and random keys representation (Bean, 1994; Norman and Bean, 1995) schemes have been used for solving JSSPs.
In indirect approach, schedule is encoded using rules. For example, in priority rule based representation, (Storer et al., 1992; Dorndorf and Pesch, 1995) chromosomes are encoded with PDRs and a schedule is constructed using a priority dispatching heuristic based on the sequence of PDRs in the chromosome. Under this scheme, GAs are used to evolve a better sequence of dispatching rules. In preference-list based representation, a chromosome is constructed using 'm' sub-chromosomes, each for one machine. A sub-chromosome does not describe the operation sequence on the machine, but it is a preference list. Thus, each machine has its own preference lists. The actual schedule is generated by simulation using the preference list when waiting queue size in front of machines is greater than one. Falkenauer and Bouffoix (1991) and Croce et al., (1995), employed preference-list based representation. Tamaki and Nishikawa (1992) proposed a disjunctive graph based representation in which chromosomes are not used to represent a schedule but used as a decision tree and the schedule is deduced from chromosomes. Dorndorf and Pesch (1995) proposed a machine based representation in which a chromosome is encoded as a sequence of machines and a schedule is constructed using a shifting bottleneck heuristic (SBH). GA is used to evolve a better sequence of machines for the SBH.

Considering the objectives of the proposed research, direct schemes are found to be more suitable. Among the direct schemes for encoding, operation based, job based and random keys representation schemes are more simple methods and the computational complexity involved in deducing a schedule from the chromosome is less compared to the other direct schemes. Further, these three schemes do not require any problem specific information. The results of the investigation done to select an appropriate encoding method to represent the schedule to solve JSSPs using GA are presented below.

5.2 OPERATION BASED REPRESENTATION

Operation-based representation scheme (Bierwirth, 1995; Gen and Cheng, 1997) is the most natural representation scheme and is highly suitable for the Pareto optimization problem because this representation covers only all feasible solutions of a JSSP (Mattfeld, 1996). Operation based representation encodes a schedule as a sequence of operations with each
gene representing one operation. All the operations of a particular job are represented by its job number and are interpreted according to the order of occurrence.

\[
\begin{align*}
3 & 10 & 5 & 3 & 0 & 8 & 4 & 3 & 2 & 3 & 1 & 2 \\
4 & 3 & 2 & 8 & 1 & 2 & 0 & 6 & 3 & 5 & 5 & 1 \\
3 & 2 & 0 & 10 & 2 & 7 & 4 & 4 & 5 & 9 & 1 & 10 \\
5 & 8 & 1 & 6 & 3 & 5 & 2 & 7 & 4 & 8 & 0 & 3 \\
2 & 5 & 1 & 10 & 3 & 2 & 0 & 1 & 5 & 8 & 4 & 8 \\
2 & 7 & 3 & 10 & 1 & 7 & 4 & 8 & 5 & 1 & 0 & 7
\end{align*}
\]

Figure 5.2. Problem data for a 6 job 6 machine problem TA01 generated using Taillard's (1993) algorithm

The problem data format used in the present work is shown in Figure 5.2. It consists of a number of rows equal to number of jobs, one line of description for each job. Each line lists a number of pairs of data equal to number of operations or machines with each pair representing the machine number and processing time for each operation. For example, the first element of \( j^{th} \) pair in the \( i^{th} \) row represents the machine using which \( j^{th} \) operation of \( i^{th} \) job is to be processed for the duration specified in the second element of the \( j^{th} \) pair. All jobs and machines are numbered starting with zero. In the above example, the first operation of job 1 is to be processed using machine 3 for a duration of 10 minutes.

\[
(3 \ 4 \ 4 \ 5 \ 2 \ 6 \ 4 \ 3 \ 3 \ 1 \ 4 \ 1 \ 3 \ 6 \ 5 \ 6 \ 5 \ 2 \ 6 \ 1 \ 2 \ 6 \ 5 \ 1 \ 5 \ 3 \ 2 \ 1 \ 2 \ 3 \ 1 \ 2 \ 6 \ 4 \ 5 \ 4)
\]

Figure 5.3. Randomly generated chromosome for a 6 job 6 machine problem

A randomly generated chromosome for a six-job six-machine job shop scheduling problem is shown in Figure 5.3. Reading from left to right in Figure 5.3, the first integer 3 refers to the first operation of job 3 and the second one is 4 which refers to the first operation of job 4. The third one is again 4 which refers now to the second operation of the job 4 and so on. This kind of decoding the representation from left to right is called semi-active decoding (Mattfeld, 1996). Sometimes chromosomes are generated either by using PDRs or any other heuristics.
5.2.1 Schedule Builder

Schedule builder is an algorithm used to construct a feasible schedule using the operation order given by the chromosome and problem data. A schedule builder algorithm has been developed to create the schedule for JSSPs in the order of operations present in the chromosome. The chromosome in Figure 5.3 results in the schedule represented in Figure 5.4 for the JSSP data given in Figure 5.2.

\[(M_1,M_2,M_3,M_4,M_5,M_6 \text{ are machines and } 1,2,3,4,5,6 \text{ are jobs.})\]

\[
\begin{align*}
M_1 & \quad 0033333333330011111111000000000500444022222206666666 \\
M_2 & \quad 0000000444445555555555000006666662233333333331100 \\
M_3 & \quad 5555566666633333344444422222222111100000000000000 \\
M_4 & \quad 33111111111104444466666666665500000000000022222200 \\
M_5 & \quad 222000000000000000033331114444444400666666655555555 \\
M_6 & \quad 44444444400011100000000333333335555555000060000200
\end{align*}
\]

Figure 5.4. Schedule obtained from the operation based chromosome representation.

In the Gantt chart like display shown in Figure 5.4, zero indicates machine idleness. It is to be noted that, during schedule construction no machine is kept idle at a time when it could begin processing some operation.

5.3 JOB BASED REPRESENTATION

Job based representation scheme is used by Holsapple et al. (1993) to generate schedule in Flexible Manufacturing Systems (FMS). Under this, chromosome consists of a list of 'n' jobs and a schedule is constructed according to the sequence of jobs. For a given sequence of jobs, all operations of the first job in the list are scheduled first and then the operations of the second job in the list are considered, and so on, until all operations of the jobs are scheduled. To illustrate, a randomly generated job sequence (1 0 5 4 2 3) is considered. It represents a chromosome for a six-job problem. Using this, the schedule generated by the scheduler using the problem data given in Figure 5.2 is shown in Figure 5.5.
5.4 RANDOM KEYS REPRESENTATION

Random key representation encodes a solution using a set of random numbers between 0 and 1. These values are used as sort keys to decode the schedule. Sorting the random numbers provides the job sequence. The random key representation scheme (Bean, 1994) requires no repair mechanism when genetic operators are applied. For instance, consider a chromosome (0.118 0.351 0.522 0.764 0.729 0.503). Sorting these numbers in ascending order results in the job sequence (1 2 4 6 5 3). This sequence is then used to construct schedule as explained in section 5.3.

5.5 EXPERIMENTAL INVESTIGATION

In the first stage of investigation on selection of a representation scheme for solving JSSPs, the operation-based and the job-based chromosome representations are studied using a benchmark problem proposed by Muth and Thompson (1963) and few randomly generated problems (see Appendix A) using the algorithm proposed by Taillard (1993). Performance of GAs with these two representation structures is compared in terms of the quality of the schedule generated and the computational effort required. The computational results are presented in Table 5.1. The results obtained after 100 generations.

Results (Suresh and Mohanasundaram, 2000) indicate that the GA, for solving JSSPs, performs better with the operation based representation than with the job based representation.
In the second part of the study, application of random keys representation for job shop scheduling problem is investigated (Suresh et al., 2000b) and found that the elite solution obtained is lost often and the GA is found to be not converging. This is due to the no-Lamarkian characteristic of the data structure, which does not permit inheritance by offspring from parents. Therefore, the operation-based representation scheme is used to represent JSSP in our further research.

Table 5.1. Performance comparison of representation schemes for JSSPs

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Problem</th>
<th>Operation based Representation</th>
<th>Job based Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$C_{max}$</td>
<td>Average CPU Time (Sec)</td>
</tr>
<tr>
<td>1.</td>
<td>MT06</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>TA06</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>TA02</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>TA03</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>TA04</td>
<td>61</td>
<td>8.20237</td>
</tr>
<tr>
<td>6.</td>
<td>TA05</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>TA01</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>TA07</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>TA08</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>TA09</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

5.6 INITIAL POPULATION

A good initial population is expected to improve the performance of the population based approach. Initial population can be generated by using different heuristics and this will increase the computational load. In our study, only 10% of the initial population is filled with chromosomes generated by using priority dispatching rules (PDRs). Remaining portion of the population is filled with randomly generated chromosomes. For this purpose, PDRs that
make use of process time such as shortest processing time (SPT) first and longest processing
time (LPT) first, and PDRs that make use of work content of jobs in the queue for the next
operation such as maximum work content remaining (MWKR) and least work content
remaining (LWKR) are studied.

The performance of these PDRs have been studied by constructing schedules separately
for all benchmark JSSPs using these PDRs. Chromosomes generated for certain problems by
using SPT rule are presented in Figures B1 to B12 in Appendix B. The Objective function
values of schedules constructed using different PDRs for the benchmark JSSPs under study
are presented in the Tables B1 to Table B13 in Appendix B.

The SPT rule is found to minimize mean flow time and also shows good performance
related to the makespan objective. LWKR rule minimizes mean flow time and the MWKR
rule minimizes the makespan measure. Therefore, chromosomes generated by SPT, LWKR
and MWKR are used to fill up 10% of the initial population.

5.7 THE PARETO SELECTION SCHEME

The next stage of investigation is carried out to identify a suitable method of selection.
During selection, individual chromosomes are selected from the current population according
to their survival probability to create offspring for the next generation. The goal of a multi­
objective GA is to generate more numbers of non-dominated solutions close to the Pareto
solutions of the problem. In addition, diversity of the non-dominated solution set is to be
increased to maintain uniform spread among the non-dominated solution set.

It is noted from the literature that MMOGLS proposed by Ishibuchi et al. (1998;
2003), NPGA2 proposed by Erickson et al. (2001), SPEA2 proposed by Zitzler et al. (2001)
and NSGA II proposed by Deb et al. (2002) are the state-of-the-art multi-objective
evolutionary algorithms. The MMOGLS (Ishibuchi et al., 2003) algorithm has been tested
using scheduling problems. All other algorithms have been tested using continuous test
functions.
Ishibuchi et al. (2003) used a weighted sum of multiple objectives as a fitness function for selection and thereby transformed the multi-objective problem into a single objective one. The randomly generated weights aim to specify different search directions towards the Pareto optimal front. By changing the weights, the algorithm has been directed to search in non-convex fronts and produced better results.

Erickson et al. (2001) proposed Niched-Pareto optimization (NPGA2) algorithm using Pareto ranking with tournament selection. Zitzler et al. (2001) proposed a Strength Pareto Evolutionary Algorithm (SPEA2) in which the fitness of each individual is computed based on the number of individuals that dominate it and the number of individuals by which it is dominated. Deb et al. (2002) employed tournament selection method and they preferred solutions with a lower non-domination rank value in their NSGA-II algorithm during comparison.

In this research, two benchmarking procedures, namely, MMOGLS and NSGA II are considered for the purpose of comparison to evaluate the performance of the proposed algorithm.

5.8 NON-DOMINATION RANKING

Complexity involved in non-dominated sorting and ranking procedure presently used by researchers is more. There is no efficient algorithm to check for non-dominance in a set of feasible solutions. Among the ranking procedure, the non-domination ranking procedure proposed by Deb et al., (2002) is better. It is to be noted that the storage requirement is $O(PS^2)$. In this section, a simple and fast non-domination ranking procedure is proposed.

The advantages of the new procedure are listed below.

1. The complexity is $O(K \times PS^2)$
2. Storage requirement is $O(PS)$
3. Only one variable is used that is domination count
4. Reduced CPU time
5.8.1 Easy and Fast Non-domination Ranking

In the proposed sorting and ranking procedure, every chromosome is evaluated to determine various objective functions \( f(x) = \{f_1(x), f_2(x), \ldots, f_K(x)\} \). To determine non-domination rank of individuals, every solution is compared with other members in the population. A solution \( x^1 \in PS \) is said to dominate the other solution \( x^2 \in PS \), if both the following conditions are true (Deb, 2002).

i) The solution \( x^1 \in PS \) is no worse than \( x^2 \in PS \) in all objectives.

ii) The solution \( x^1 \in PS \) is strictly better than \( x^2 \in PS \) in at least one objective.

Each solution in the population is compared with every other solution in the population for non-domination. When both the conditions are satisfied, \( x^2 \) is called a dominated solution and \( x^1 \) a nondominated solution. If any of the above condition is violated, the solution \( x^1 \) does not dominate the solution \( x^2 \).

In this proposed approach, an index namely non-domination count is used. Initially domination count is assigned to be zero to all chromosomes. Every solution \( i \) in the population is compared with every other solution \( j \) in the population for non-domination. If \( (i \) dominates \( j) \), then domination count of \( j \) is incremented. Based on the domination count, various fronts are formed. This algorithm requires no additional memory variables and hence memory storage.

It is clear that after all comparisons, members belonging to the first front will have their domination count zero. First, members with zero domination count are assigned rank one. Leaving the first front, members with least value of domination count are assigned rank two and the procedure is repeated until all members are assigned non-domination ranks. The pseudo code of the proposed ranking procedure is presented in Figure 5.6.
for $i = 1 \text{ to } PS$

$n_i = 0$

for $i = 1 \text{ to } PS$

for $j = 1 \text{ to } PS$

if ($i \neq j$)

if (if $i$ dominates $j$)

then ($n_j = n_j + 1$)

$t \rightarrow 1$

$q = 0$

do while ($q < PS$)

$F_t = \{ \}$

$c = 0$

for $i = 1 \text{ to } PS$

if ($n_i = q$)

$F_t = \{ F_t \} + i$

$c = c + 1$

$F_t = \{ F_t \} + i$

$t = t + 1$

increment $t$

$q = q + c$

Figure 5.6 Easy and fast non-domination ranking algorithm

5.8.2 Numerical Illustration

The proposed sorting procedure is explained with an example. The population members obtained after 1000 generations, by GA employing the Pareto selection, for the LA01 benchmark JSSP are presented along with their objective function values in Table 5.2. The population size chosen is 50. Domination count $n_i$ is determined for all chromosomes. Based on the count, non-domination rank is assigned. Lesser is the count, lesser the rank.
Table 5.2. Proposed non-domination rank assignment with population size PS=50.

<table>
<thead>
<tr>
<th>SL. No</th>
<th>$C_{\text{max}}$</th>
<th>$\bar{F}$</th>
<th>$n_i$</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>666</td>
<td>547.3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>666</td>
<td>550.2</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>666</td>
<td>550.7</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>666</td>
<td>553.9</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>666</td>
<td>562.5</td>
<td>43</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>668</td>
<td>541</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>669</td>
<td>543.4</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>670</td>
<td>552.1</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>671</td>
<td>550.4</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>671</td>
<td>550.7</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>672</td>
<td>536.1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>672</td>
<td>543.2</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>672</td>
<td>544.2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>672</td>
<td>549.9</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>673</td>
<td>544.9</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>680</td>
<td>540.4</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>682</td>
<td>546.1</td>
<td>43</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>690</td>
<td>532.4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>690</td>
<td>543.2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>690</td>
<td>544.2</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>693</td>
<td>542</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>694</td>
<td>543.4</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td>695</td>
<td>541.2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>697</td>
<td>534.1</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>699</td>
<td>534.4</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>702</td>
<td>542.1</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>705</td>
<td>528.5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>706</td>
<td>543.1</td>
<td>43</td>
<td>5</td>
</tr>
<tr>
<td>29</td>
<td>709</td>
<td>528.3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>710</td>
<td>520.5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>710</td>
<td>527.6</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>32</td>
<td>714</td>
<td>522.4</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>33</td>
<td>715</td>
<td>525.1</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>715</td>
<td>537.4</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>717</td>
<td>540.3</td>
<td>43</td>
<td>5</td>
</tr>
</tbody>
</table>
Members belonging to various fronts are plotted in Figure 5.7. Various non-domination fronts \( (F_1, F_2, \ldots) \) are sorted based on their domination count.

![Graph showing successive non-dominated fronts obtained for the LA01 benchmark JSSP using PLSGA after 1000 generations](image)
5.8.3 Experimental Evaluation of Ranking Procedure using Continuous Test Functions

Testing of the speed of the proposed ranking procedure is done by comparing its performance with the ranking procedure used in NSGA II. It is done by solving continuous test functions proposed in the literature (Deb et al., 2002; Coello et al., 2002). Names of the test functions are SCH (Schaffer, 1987), FON (Fonseca and Fleming, 1998) and POL (Poloni, 1997). Objective functions of the test problems and their constraints are presented in Table 5.3. These problems are solved using NSGA II using first by the sorting procedure given by Deb et al. (2002) and then by using the easy and the fast ranking procedure proposed in this research. Both are run for a maximum number of 25000 function evaluations. Population members obtained for problems SCH, FON and POL are plotted in Figures 5.8 to 5.10. CPU time taken by these two algorithms is reported in Table 5.4 for different population size.

Table 5.3. Test problems used for testing the ranking procedure

<table>
<thead>
<tr>
<th>Problem</th>
<th>Objective Function</th>
<th>Variable bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCH</td>
<td>( f_1(x) = x^2 )  ( f_2(x) = (x - 2)^2 )</td>
<td>([-10^3, 10^3])</td>
</tr>
<tr>
<td>FON</td>
<td>( f_1(x) = 1 - \exp\left(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})^2\right) ) ( f_2(x) = 1 - \exp\left(-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}})^2\right) )</td>
<td>([-4.4])</td>
</tr>
<tr>
<td>POL</td>
<td>( f_1(x) =</td>
<td>1 + (A_1 - B_1)^2 + (A_2 - B_2)^2</td>
</tr>
</tbody>
</table>
PhD thesis on “An investigation on the application of metaheuristics to job shop scheduling problems with multiple objectives”

Problem SCH

Figure 5.8 Population members obtained by NSGA II using the easy and fast ranking procedure for the test suite SCH

Problem: FON

Figure 5.9 Population members obtained by NSGA II using the easy and fast ranking procedure for the test suite FON
Figure 5.10 Population members obtained by NSGA II using the easy and fast ranking procedure for the test suite POL

Table 5.4. CPU time taken by NSGA ranking and the proposed ranking procedure

<table>
<thead>
<tr>
<th>Problem</th>
<th>Population size</th>
<th>CPU Time (Sec)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NSGA II Ranking</td>
<td>Proposed Ranking</td>
<td></td>
</tr>
<tr>
<td>SCH</td>
<td>10</td>
<td>3.4</td>
<td>3.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>70.45</td>
<td>66.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1308.5</td>
<td>1170.14</td>
<td></td>
</tr>
<tr>
<td>FON</td>
<td>10</td>
<td>7.58</td>
<td>7.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>108.04</td>
<td>104.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1399.8</td>
<td>1294.3</td>
<td></td>
</tr>
<tr>
<td>POL</td>
<td>10</td>
<td>5.65</td>
<td>5.61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>90.37</td>
<td>87.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1325.7</td>
<td>1229.1</td>
<td></td>
</tr>
</tbody>
</table>

It is to be noted that computational time is less with the proposed procedure. From the data presented in Table 5.4, it is evident that as the population size grows, the improvement is more significant.
5.9 KEEPING THE DIVERSITY

It has already been stated that the diversity among the non-dominated front is very important. The crowded comparison operator proposed by Deb et al. (2002) is a simple approach to maintain diversity among solutions of the non-dominated front. It is an absolute measure of density of the neighbourhood covering the solution under consideration with respect to various objectives in the same non-dominated front. The performance of the above approach is further improved and the improved version of crowded comparison operator is presented in the following section. Advantage of the new operator is also explained with numerical illustration.

5.9.1 Numerical Illustration

Computation of the crowding comparison operator value is explained using three non-dominated solutions A, B and C as shown in Figure 5.11. The crowding distance of the solution C as per Deb et al. (2002) is the average side length of the cuboid shown with a dashed box. For that matter, it is to be noted that the crowding distance of any member between A and B like C₁, C₂ or C₃, shown in Figure 5.12 is the same.

![Figure 5.11. Cuboid construction as per Deb et al. (2002) to determine the crowding distance](image)
It is clear that $C_2$ is more preferable than $C_1$ and $C_3$. Because, $C_1$ has a close neighbour $A$ and $C_3$ has a close neighbour $B$. Instead of using the perimeter of the cuboid formed by the nearest neighbours, perimeter of the cuboid formed with the solution itself in one vertex and its upstream or downstream neighbour in the other vertex can be used to compute the crowding distance.
The new crowding distance computation procedure is presented in Figure 5.14.

```plaintext
for i = 1 to PS
    for j = 1 to K
        Pa' = 0; Pb' = 0;  
        where \( Pa'_i = \sum_{j=1}^{K} Pa'_j \) and \( Pb'_i = \sum_{j=1}^{K} Pb'_j \)

        \( Pa'_j \) refers to half the perimeter of a square formed with \( i^{th} \) solution in one vertex and its adjacent solution with a higher \( j^{th} \) objective function value in the opposite vertex.

        \( Pb'_j \) refers to half the perimeter of a square formed with \( i^{th} \) solution in one vertex and its adjacent solution with a lower \( j^{th} \) objective function value in the opposite vertex.

for j = 1 to K
    To initialize neighbours of the extreme solutions
    \( f_j^0 = f_j^1 - 2*(f_j^{\text{max}} - f_j^{\text{min}}) \)
    \( f_j^{PS+1} = f_j^{PS} + 2*(f_j^{\text{max}} - f_j^{\text{min}}) \)

for j = 1 to K
    sort \((PS, j)\)  
    Sort population members in the ascending order of \( j^{\text{th}} \) objective
    for i = 1 to PS
        \( Pa'_i = Pa'_i + (f_j^i - f_j^{(i-1)})/((f_j^{\text{max}} - f_j^{\text{min}})) \)
        \( Pb'_i = Pb'_i + (f_j^{i+1} - f_j^i)/((f_j^{\text{max}} - f_j^{\text{min}})) \)

for i = 1 to PS
    \( DS_i = Pa'_i * Pb'_i \), \( DS_i \rightarrow \text{refers to DS value of } i^{\text{th}} \text{ solution} \)

Figure 5.14 Pseudo code for the proposed diversity measure computation for \( K \) objective problem
```
Crowding distances for the solutions $C_1$, $C_2$, and $C_3$ marked in Figure 5.13 is computed as per Deb et al. (2002) and the proposed approach. The results are presented in Tables 5.5 to 5.7. The new approach is termed as $DS$ approach (Due to Deb et al. and Suresh et al.) and the new crowding distance is called $DS$ value. This approach is suitable for any number of objectives. It is to be noted that, to compute $DS$ value, $2K$ numbers of squares are to be constructed for every solution. A solution with smaller value of this measure is more crowded by other solutions.

Table 5.5 Crowding distance calculation for the non-dominated solution $C_1$

<table>
<thead>
<tr>
<th></th>
<th>f1</th>
<th>f2</th>
<th>NSGA II $i_{distance}$</th>
<th>Proposed $P_a$</th>
<th>$P_b$</th>
<th>$DS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1200</td>
<td>12450</td>
<td>4.513</td>
<td>4.000</td>
<td>0.513</td>
<td>2.051</td>
</tr>
<tr>
<td>C1</td>
<td>1248</td>
<td>12375</td>
<td><strong>2.000</strong></td>
<td>0.513</td>
<td>1.487</td>
<td><strong>0.763</strong></td>
</tr>
<tr>
<td>B</td>
<td>1400</td>
<td>12175</td>
<td>5.487</td>
<td>1.487</td>
<td>4.000</td>
<td>5.949</td>
</tr>
<tr>
<td></td>
<td>1800</td>
<td>11625</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6 Crowding distance calculation for the non-dominated solution $C_2$

<table>
<thead>
<tr>
<th></th>
<th>f1</th>
<th>f2</th>
<th>NSGA II $i_{distance}$</th>
<th>Proposed $P_a$</th>
<th>$P_b$</th>
<th>$DS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1200</td>
<td>12450</td>
<td>4.991</td>
<td>4.000</td>
<td>0.991</td>
<td>3.964</td>
</tr>
<tr>
<td>C2</td>
<td>1300</td>
<td>12315</td>
<td><strong>2.000</strong></td>
<td>0.991</td>
<td>1.009</td>
<td><strong>1.000</strong></td>
</tr>
<tr>
<td>B</td>
<td>1400</td>
<td>12175</td>
<td>5.009</td>
<td>1.009</td>
<td>4.000</td>
<td>4.036</td>
</tr>
<tr>
<td></td>
<td>1800</td>
<td>11625</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.7 Crowding distance calculation for the non-dominated solution $C_3$

<table>
<thead>
<tr>
<th></th>
<th>f1</th>
<th>f2</th>
<th>NSGA II $i_{distance}$</th>
<th>Proposed $P_a$</th>
<th>$P_b$</th>
<th>$DS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1200</td>
<td>12450</td>
<td>5.477</td>
<td>4.000</td>
<td>1.477</td>
<td>5.909</td>
</tr>
<tr>
<td>C3</td>
<td>1350</td>
<td>12250</td>
<td><strong>2.000</strong></td>
<td>1.477</td>
<td>0.523</td>
<td><strong>0.772</strong></td>
</tr>
<tr>
<td>B</td>
<td>1400</td>
<td>12175</td>
<td>4.523</td>
<td>0.523</td>
<td>4.000</td>
<td>2.091</td>
</tr>
<tr>
<td></td>
<td>1800</td>
<td>11625</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is to be noted that, the locations \( C_1 \) and \( C_3 \) has lesser crowding distance as per the new procedure. This indicates that they are more crowded.

Maintaining diversity is more important when large number of non-dominated solutions is generated during the search process. This happens mostly while solving continuous optimization problems. For the discrete optimization problems such as scheduling, non-dominated solutions are not available at very close intervals. Therefore, usefulness of the crowding distance measure in such cases require further study.

### 5.10 TOURNAMENT SELECTION

Tournament selection has been employed in the present work with non-domination measure as the primary selection criterion. Between chromosomes with different ranks, solution with lower rank is preferred. If both solutions belong to the same front, solution with higher DS value is preferred. Lower rank refers to the closeness of the particular front to the Pareto front. Higher DS value indicates that the solution is present in a lesser crowded area. Non-domination rank of individuals in the population and DS values for members belonging to various fronts are computed prior to the selection process. During selection, entire population is shuffled \( TS \) times and \( \frac{PS}{TS} \) number of groups are randomly selected every time before shuffling. Thus, from the available \( PS \) number of groups, \( PS \) number of chromosomes are selected.

With too much selection pressure, genetic search leads to premature convergence; with less pressure, evolution will be slower. Tournament size determines the selection pressure. Lowest possible tournament size avoids premature convergence of the algorithm. This is clear from the application of binary tournament selection in almost all the state-of-the-art multi-objective algorithms. However, with increased selection pressure promising fronts are found to survive better. In the proposed approach, tournament size is taken as five.

### 5.11 CROSSOVER OPERATOR

From the selected population, different pairs of chromosomes are selected, in the order of presence for crossover with a specified crossover probability. In this section, performance of
three popular crossover methods suitable for the operation based representation schemes, are studied. Crossover procedures selected for study are single point crossover (Ishibuchi et al., 1996), two point crossover (Ishibuchi and Murata, 1998) and partial schedule exchange crossover (Gen et al., 1994). These schemes are explained in sections 5.9.1 to 5.93 with illustrations.

5.11.1 Single Point Crossover

In single point crossover following steps are involved.

Step1: Start with Parent 1 and select a cut point along parent1 at random.

Step2: Produce an offspring1 by copying first part of parent1.

Step3: Delete all the corresponding operations from the parent 2, which are already in the first part of parent1. The resulting sequence contains the operations the offspring1 needs.

Step4: Add the remaining operations in parent 2 into the offspring1 after the first part from left to right according to the order of the sequence.

Step5: Repeat the above steps with parent 2 to produce the offspring 2.

The above procedure is explained below in Figure 5.15

![Figure 5.15 Single point crossover](image-url)
5.11.2 Two Point Crossover

Two point crossover for the operation based representation involves the following steps.

Step 1: Select a substring \((a-b)\) from parent1 at random.

Step 2: Produce a proto-child by copying the substring1 into positions corresponding to those in the parent.

Step 3: Delete all the symbols from the parent2, which are already in the substring1. The resulting sequence in parent2 contains the operations the proto-child needs.

Step 4: Place remaining operations in parent2 into unfixed positions of the proto-child from left to right according to the order of the sequence to produce the offspring1.

Step 5: Repeat the above steps with parent 2 to produce the offspring2.

The above procedure is explained schematically in Figure 5.16.

![Two point crossover diagram](image-url)

Figure 5.16 Two point crossover
5.11.3 Partial Schedule Exchange Crossover

Partial schedule exchange crossover for the operation based encoding involves the following operations.

Step 1: Identify a partial schedule in one parent randomly and in the other parent accordingly.

Step 2: Exchange the partial schedules to generate proto-offspring.

Step 3: Determine the gene missed and exceeded for the proto-offspring.

Step 4: Legalize offspring by deleting genes exceeded and adding genes missed in the order of appearance as explained in the Figure 5.14.

The above procedure is explained schematically in Figure 5.17.

![Figure 5.17 Partial schedule exchange crossover](image-url)
Performance of these crossover operators, when used with operation based representation scheme to solve multi-objective JSSPs, is analysed (Suresh et al., 2001a) by solving some of the benchmark JSSPs. The non-dominated front obtained by the two-point crossover operator is found to have contributed more number of solutions to the net front. Experimental results obtained using the above three crossover operators for the popular MT06 benchmark problem are presented in Table 5.8. Therefore, the two-point crossover operator is used in our further study.

Table 5.8 Non-dominated solutions obtained using various crossover operators for the popular mt06 benchmark problem

<table>
<thead>
<tr>
<th></th>
<th>TPX</th>
<th>PSE</th>
<th>SPX</th>
<th>COMBINED FRONT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{max}}$</td>
<td>$\overline{F}$</td>
<td>$C_{\text{max}}$</td>
<td>$\overline{F}$</td>
<td>$C_{\text{max}}$</td>
</tr>
<tr>
<td>55</td>
<td>50.8*</td>
<td>57</td>
<td>49.5*</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>46.7*</td>
<td>58</td>
<td>48.8</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>46.5*</td>
<td>60</td>
<td>46.7</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>46.5</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>47.83</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>71</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>46.66</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.12 MUTATION

The role of mutation in GA is to combat premature convergence. The mutation rate $p_m$ determines the probability of altering a chromosome (generating a modified schedule) by means of mutation. Three simple mutation operators (Mattfeld, 1995) are considered in this section. They are: order based mutation (OBM), position based mutation (PBM) and swap based mutation (SBM). In the swap-based mutation, one locus at random is identified and operations are exchanged with adjacent one. It is found that the SBM is inferior in terms of generating a modified schedule from the parent schedule and therefore the PBM and the OBM are considered further.

An effective local search procedure has been applied to improve the performance of the search process. This technique is explained in section 5.13. It works based on the PBM
procedure to identify neighbourhood solutions. Therefore, the order-based mutation is employed in the present work. The order-based mutation is also called as random swap mutation. It is done by selecting two loci (positions) in a chromosome at random and exchanging their alleles (corresponding operations).

A mutation rate of $P_m=1.0$ is chosen to permit swapping of $(n \times m)$ number of operations among $(PS \times n \times m)$ number operations present in the population.

5.13 HYBRIDIZATION WITH LOCAL SEARCH

The performance of EAs is greatly improved when they are hybridized with other techniques (Dimopoulos and Zalzaia, 2000). Zhou et al. (2001) claim that the efficiency of the traditional GA can be improved using heuristic rules and local search techniques. It has been reported (Ishibuchi and Murata, 1998) that by incorporating local search mechanism performance of the GA was largely improved. Knowles (2002) observed that, much of the work in the GA literature exhibit poor performance and simple GAs require additional feature such as local search in order to outperform simpler technique like simulated annealing. In the present work, we are interested in the use of local search for the following reasons.

1. It is found that non-dominated solutions present in a population get more number of their copies reproduced into the population due to Pareto selection. Such behaviour makes the genetic operators inefficient. To overcome this, local search is applied to generate a diverse population.

2. When GAs are applied to solve multi-objective optimization problems, selection and crossover operators sample the search space with solutions. GA combined with local search mechanism is considered useful in exploring the sample search space identified by the genetic operators.

GA is run first to generate new population and the local search has the advantage of improving the rank of every individual in the population. To be clear, genetic search is useful
to approach the Pareto front and the local search is employed afterwards to assist the multi-objective GA program to identify more members of the Pareto front.

A local search mechanism based on the successful performance of the insertion scheme of Sridhar and Rajendran (1993) is used in this section. They used the insertion scheme as a perturbation method in SA for solving FSSPs. A similar method has also been used as a mutation operator namely position based mutation by Mattfeld (1995) in his GA for solving JSSPs. The local search method used in this section is termed as segment random insertion (SRI) scheme which is explained below.

Step 1: Let \([211213233]\) be a parent chromosome selected for local search from the population. Select randomly a segment consisting of \(l_s\) number of operations. In this case the segment \((213)\) is selected.

\[
\text{Segment: } 2 \quad 1 \quad 1 \quad 2 \quad 1 \quad 3 \quad 2 \quad 3 \quad 3
\]

Step 2: Insert each job in the segment at any position randomly to its left and right to generate \(2^{\frac{n}{l_s}}\) number of neighbours as given in Table 5.9.

<table>
<thead>
<tr>
<th>Job No.</th>
<th>Point of insertion (Left)</th>
<th>Neighbours Obtained</th>
<th>Point of insertion (Right)</th>
<th>Neighbours Obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>221113233</td>
<td>9</td>
<td>211132332</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>121123233</td>
<td>7</td>
<td>211232133</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>321121233</td>
<td>8</td>
<td>211212333</td>
</tr>
</tbody>
</table>

Step 3: Delete dominated chromosomes within the neighbourhood set. After deleting dominated chromosomes, if more than one remains randomly select one as the best neighbour.

Step 4: If the best neighbour dominates the parent chromosome, then replace the parent with the best neighbour and goto step 7.
Step 5: if the parent chromosome dominates the best neighbour, accept the best neighbour with a probability $P_a$.

$$P_a = \exp\left(-\frac{\psi}{\gamma}\right) \quad eqn. \ldots \ldots \text{5.1}$$

where $\psi = \sum_{i=1}^{k} \left[ (f_i^* - f_i^*) / (f_i^* - f_i^*) \right]$ \quad eqn. \ldots \ldots \text{5.2}$

Step 6: If both are non-dominated solutions, replace the parent with the neighbour with the probability of 50%.

Step 7: Repeat steps through 1 and 6 for other chromosomes in the population.

In the present work, a segment or sub chromosome of fixed size $l_s$ is selected randomly and each member of the segment are randomly inserted to the right or left side of the segment to generate $(2 \times l_s)$ number of neighbours. To reduce the time spent in local search, the following measures are taken.

1. The size of the segment is fixed as 5 irrespective of the problem size
2. The probability of local search for the population members is taken as 0.5

### 5.13.1 Effect of the Local Search Mechanism: An Investigation

Performance of SRI scheme is examined first with NSGA II. For this, the famous benchmark job shop scheduling problem namely FT10 is evaluated using the NSGA II algorithm and the population obtained is shown in Figure 5.19. Afterwards, local search using SRI scheme is applied to assist NSGA II for the same problem and the final population is presented in Figure C1 in appendix C.

In the NSGA II approach employed in this section, NSGA selection is employed and the crowded comparison operator suggested by Deb et al., (2002) is used to maintain diversity. However, the operation based representation is employed for coding the solution. Along with this, the two-point crossover operator and the order based mutation operator described earlier are used. This structure of NSGA II is referred in this section as ‘heuristic1’ and the same structure of NSGA II with SRI local search method is referred as ‘heuristic2’.
Further, to evaluate the effectiveness of the local search more number of selected benchmark JSSPs are solved using both heuristic 1 and heuristic 2. The problems selected for solving are hard to solve (Mattfeld, 1996; Jain and Meeran, 1999) even with single objective measure. The final population obtained after evaluating the specified number of solutions for the tough benchmark JSSPs, namely, FT10, LA21, LA24, LA25, LA27, LA29, and LA38 using both heuristic 1 and heuristic 2 are presented in Figures C1 to C14 in Appendix C.

A general observation from the results presented above is that, the local search mechanism is very effective in maintaining population diversity and thereby it assists the genetic search mechanism to harvest solutions closer to the Pareto front every time. Therefore, it is worth implementing the local search during every generation. As stated earlier, to strike a balance between genetic and local search (Ishibuchi et al., 2003), selected number of chromosomes undergo local search. About 50% of the population members are randomly selected for local search.

Further, the effectiveness of the local search technique is known better by solving tough benchmark JSSPs. It has been observed that when the ratio \((n + m)\) is close to one, the quality of the non-dominated fronts obtained by the heuristics is poor. This is especially true for the benchmark JSSPs such as YN01 to YN04 where \(n=20, m=20\) and SWV06 to SWV10 where \(n=20, m=15\). It has also been reported (Ramudhin and Marier, 1996) that coefficient of workload increases when \((n > m)\) making it easier to select the bottleneck machine, thus reducing the possibility of being trapped in local optima. When benchmark JSSPs such as FT20 is solved using heuristic 1, a non-dominated front with better spread and less percent deviation of extreme solutions from the respective optimal solutions is obtained. This may be due to \(n = 4m\). But, the quality of the front obtained using the heuristic 2 is found to be still better than that of the former algorithm. Non-dominated fronts obtained by these heuristics are shown in Figure 5.18. For easy grouping of the non-dominated solutions, a line is drawn to connect them.
In continuation of the above study, the benchmark JSSPs proposed by Adams et al. (1988) such as ABZ5 (n=10, m=10) and ABZ6 (n=10, m=10) have been solved using both the heuristics. Non-dominated fronts obtained by both the heuristics are presented in Figures 5.19 and 5.20. Interestingly, the results obtained are comparable. However, the results obtained for the problems ABZ7 to ABZ9 (n=20, m=15) shows that contribution of the local search mechanism is more in obtaining a good quality non-dominated front in terms of the quality measures discussed in the chapter 4. The results for the problems ABZ7 to ABZ9 are presented in Figures 5.21 to 5.23.

Figure 5.18. Non-dominated fronts obtained by heuristic1 and heuristic2 for the problem FT20 (n=20, m=5)
PhD thesis on “An investigation on the application of metaheuristics to job shop scheduling problems with multiple objectives”

Figure 5.19. Non-dominated fronts obtained by heuristic 1 and heuristic 2 for the problem ABZ5 (n=10, m=10)

Figure 5.20. Non-dominated solutions obtained by heuristic 1 and 2 for the problem ABZ6 (n=10, m=10)
PhD thesis on "An investigation on the application of metaheuristics to job shop scheduling problems with multiple objectives"

Figure 5.21. Non-dominated solutions obtained by heuristic 1 and 2 for the problem ABZ7 (n=10, m=10)

Figure 5.22. Non-dominated solutions obtained by heuristic 1 and 2 for the problem ABZ8 (n=20, m=15)
5.14 ELITISM

In the case of multi-objective optimization using GAs, preserving the non-dominated solutions found during the genetic search process is important as the elite solutions may sometimes be lost during the genetic evolution process due to the disruptive effects of genetic operators. Bagchi (1999) and Zitzler (1999) have maintained a separate elite population consisting of non-dominated solutions copied from the population at the end of every generation. Deb et al. (2002) ensured elitism by always combining the parent population with the child population before selection.

In the present work, it is proposed to store the non-dominated solutions or the elite solutions obtained during the search process separately, and it is called an ‘archive’. This elite population is updated frequently. Updating involves the following tasks.
1. Copying non-dominated solutions obtained by the algorithm into the archive

2. Comparing archive members after certain number of generations to identify dominated members and deleting them from the archive.

This has been implemented using the linear linked list concept in C language. The list grow in size after every generation. When the list size becomes large, every member of the archive is compared with every other member of the archive to identify non-dominated members. After the comparison, dominated members are removed from the list.

5.15 THE PARETO-LOCAL SEARCH GENETIC ALGORITHM: STEP BY STEP PROCEDURE

In the last few sections, various concepts of the proposed metaheuristic are discussed. These features are summarized in this section. They are:

1. Operation based coding is used to represent chromosomes

2. Ten percentage of the initial population is filled with chromosomes generated by simulation using PDRs such as SPT, LWKR and MWKR

3. Pareto selection is employed with non-domination rank as the primary criterion and the diversity measure as the secondary criterion

4. Two point crossover and random swap mutation are applied

5. Local search is applied to randomly selected population members

6. Archive is created to preserve elite solutions

With all features described above, a systematic procedure of the population based metaheuristic for solving JSSPs with multiple objectives namely the Pareto local search genetic algorithm (PLSGA) is presented below.
The step by step procedure of PLSGA.

Step 1: Read problem data

Step 2: Generate initial population. 90% of the population is generated randomly and the remaining 10% using PDRs.

Step 3: Evaluate the objective function values for all chromosomes and create an archive

Step 4: Repeat steps 5 to 11 until $n \times m \times 10000$ number of solutions are generated and evaluated

Step 5: Do non-domination ranking of the population using the easy and fast ranking procedure

Step 6: Compute crowding distance measure (DS) for all chromosomes

Step 7: Copy members of the first front into the archive and update the archive, if required

Step 8: Select PS number of chromosomes using tournament selection ($TS = 2$) procedure (section 5.10) with non-domination rank as the primary criterion and DS value as the secondary criterion

Step 9: Apply two-point crossover to selected chromosomes with the crossover probability $P_c$.

Step 10: Apply order based mutation to chromosomes with the mutation probability $P_m$.

Step 11: Apply the local search technique SRI to population members with the probability $P_{LS}$

Step 12: Output archive members.

5.16 COMPUTATIONAL TESTING OF PLSGA

The proposed multi-objective metaheuristic approach and the other benchmarking procedures described in this section have been coded in C language and the programs are run on AMD Athlon XP 2000 system. A set of benchmark JSSPs are solved with the objective of minimizing the makespan and the mean flow time. Total number of solutions evaluated is $n \times m \times 10000$ for all experiments. Non-dominated fronts obtained for different JSSPs are presented in Tables C1 to C16 in Appendix C.
PhD thesis on “An investigation on the application of metaheuristics to job shop scheduling problems with multiple objectives”

Mean percent deviation (MPD) of the extreme solutions in the non-dominated fronts obtained by PLSGA from the best upper bound reported in the literature for the benchmark job shop scheduling problems under study are computed and presented in Tables C17 to C21.

Performance of three multi-objective algorithms is compared by solving asset of benchmark JSSPs. Two quality metrics described in Chapter 4 have been used to compare the performance of various algorithms. The performance of MMOGLS on these test problems is not as good as NSGA II and PLSGA. Between the NSGA II and MMOGLS, it is seen that the NSGA II algorithm performs better. This may be due to the use of a combined mating pool in NSGA II for selection. Detailed analysis of the full results reveals that PLSGA performs most consistently in identifying non-dominated front with more solutions.

It is to be noted that, quality of the non-dominated fronts generated for many JSSPs by these algorithms is comparable. The problems include ORB04, ORB07, FT20, LA01, LA02, LA03. This is shown in Figures 5.24 to 5.29.

Figure 5.24. Non-dominated solutions obtained by MMOGLS, NSGA II and PLSGA for the problem ORB04 (n=10, m=10)
PhD thesis on “An investigation on the application of metaheuristics to job shop scheduling problems with multiple objectives”

Figure 5.25. Non-dominated solutions obtained by MMOGLS, NSGA II and PLSGA for the problem ORB07 (n=10, m=10)

Figure 5.26. Non-dominated solutions obtained by MMOGLS, NSGA II and PLSGA for the problem FT20 (n=20, m=5)
Figure 5.27. Non-dominated solutions obtained by MMOGLS, NSGA II and PLSGA for the problem LA01 (n=10, m=5)

Figure 5.28. Non-dominated solutions obtained by MMOGLS, NSGA II and PLSGA for the problem LA02 (n=10, m=5)
5.17 CONCLUSION

In this chapter, a metaheuristic framework based on genetic algorithm (GA) approach is proposed for multi-objective optimization. In the proposed metaheuristics, a local search procedure is applied to selected population members during every generation of GA. Performance of the proposed metaheuristic was demonstrated by solving a set of benchmark JSSPs with multiple objectives. The characteristic features of the proposed metaheuristic are summarized as follows.

1. Pareto selection is used with non-domination rank as the primary criterion and the diversity measure as the secondary criterion in both local search and genetic search
2. Local search is performed on certain number of randomly selected chromosomes to improve non-domination rank of the selected chromosomes.

3. An archive is created to store non-dominated solutions discovered by the metaheuristic. At regular intervals, the archive is updated by identifying and deleting dominated solutions in the archive.

4. A simple non-domination ranking procedure is developed in this section. High performance of the ranking procedure is demonstrated in terms of speed and computer memory storage requirement.

5. Less number of non-dominated solutions are present with larger and non-regular intervals in the non-dominated fronts obtained for different JSSPs. Therefore, when discrete optimization problems such as JSSPs are solved, the need for maintaining diversity among members of the non-dominated front in the solution space is found to be less significant. However, a modified crowding comparison operator is proposed in this section for better diversity control. Effectiveness of the proposed crowding comparison operator has been shown with a numerical illustration. This measure will be useful when non-dominated members are continuously present in the non-dominated front.

The proposed metaheuristic is simple and its high performance is exhibited by comparing the quality of the results obtained with the other benchmark procedures. The net front obtained for each benchmark JSSP can be used as the ‘known Pareto front’ for the JSSP.

Performance evaluation of PLSGA is carried out by solving benchmark JSSPs using PLSGA and two other popular multi-objective optimization algorithms namely MMOGLS and NSGA II and the results are compared.
In the next chapter, another metaheuristic based on simulated annealing algorithm is proposed for solving JSSPs with multiple objectives. This is an attempt to reduce distance between the known and the true Pareto front and also to identify more non-dominated solutions in every front.