P- 1679

A FLOW SHOP SCHEDULING USING GENETIC ALGORITHM

A Project Report

Submitted by

N.Kasirathi - 71204409003

in partial fulfillment for the award of the degree
of

Master of Engineering
in
Industrial Engineering

DEPARTMENT OF MECHANICAL ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006

ANNA UNIVERSITY:: CHENNAI 600 025

APRIL – 2006

ANNA UNIVERSITY:: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report entitled **"A Flow Shop Scheduling Using Genetic Algorithm"** is the bonafide work of

N.Kasirathi - Register No. 71204409003

Who carried out the project work under my supervision.

[illegible]
Signature of the Head of the Department
Dr. T. P. MANI
HEAD OF THE DEPARTMENT

N. [illegible]
Signature of the supervisor
Dr. N. Gunasekaran
SUPERVISOR

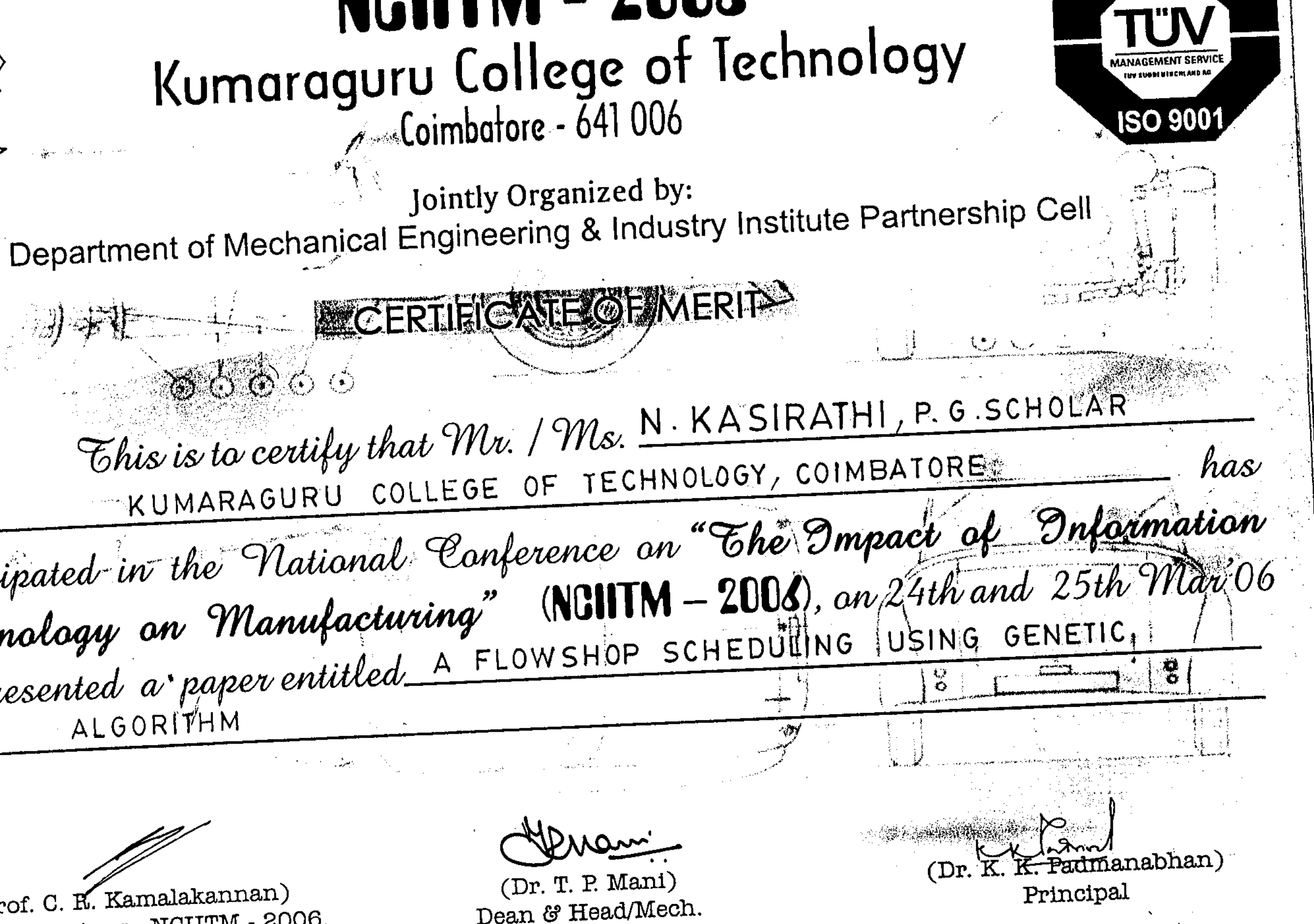
[illegible] 30/06/06
Internal Examiner

Dr. T.P. Mani
B.E., M.E., Ph.D., DML., MIE., MNIQR., MISTE.,
Dean & HoD / Dept. of Mech. Engg.
Kumaraguru College of Technology
Coimbatore - 641 006

[illegible]
External Examiner

S. R. Devadasan, Ph.D.
Professor
Production Engineering Department
PSG College of Technology
Coimbatore - 641 004, INDIA

DEPARTMENT OF MECHANICAL ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY



NCIITM - 2006

Kumaraguru College of Technology

Coimbatore - 641 006

TÜV MANAGEMENT SERVICE ISO 9001

Jointly Organized by:

Department of Mechanical Engineering & Industry Institute Partnership Cell

CERTIFICATE OF MERIT

This is to certify that Mr. / Ms. N. KASIRATHI, P.G.SCHOLAR KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE *has*

ipated in the National Conference on ***"The Impact of Information nology on Manufacturing"*** **(NCIITM – 2006)**, *on 24th and 25th Mar'06*

esented a paper entitled A FLOWSHOP SCHEDULING USING GENETIC ALGORITHM

[illegible]
rof. C. R. Kamalakannan)
NCIITM - 2006

[illegible]
(Dr. T. P. Mani)
Dean & Head/Mech.

[illegible]
(Dr. K. K. Padmanabhan)
Principal

ABSTRACT

Many industrial applications deals with the problem of improper utilization of given resources and scheduling of jobs. This project titled, "A Flow shop Scheduling Using Genetic Algorithm", explains the scheduling of jobs in a flow shop to minimize the makespan using a non-traditional technique, Genetic Algorithm.

A machine shop with 'n' number of jobs and 'm' number of machines is considered to calculate the makespan. Many sequence schedules are generated and many iterations are performed. The schedules are generated using random number generation program. The makespan calculation is based on the given processing time and batch quantity, for each schedule generated. The schedule with the minimum makespan is selected and considered as the optimum result after performing many iterations. Genetic algorithm is used for this flow shop problem due to following reasons

- One of the fast and best converging techniques when the search space is large.
- It provides better solutions when compared to other search procedures.

Genetic algorithm is implemented using C language since numbers of iterations are more. One could calculate the makespan even though the number of possible sequence schedules is (n!)^m within few minutes. In this project work, the number possible sequences are (10!)^2. An optimum result of minimum makespan is obtained at 20th iteration for a case study problem.

ஆய்வுச் சுருக்கம்

இன்றைய காலகட்டத்தில், தொழிற்சாலைகளின் முக்கியமான கேள்விக் குறிகளாக இருப்பவை, கொடுக்கப்பட்டுள்ள வேலைகளை சரியான முறையில் காலமுறைப்படுத்தாதும், சரியான முறையில் வேலைக்குத் தேவையான ஆவணங்களை உபயோகப்படுத்தாததும் ஆகும். "ஜெனிடிக் வடிவமைப்பு" என்ற புதுமையான தொழிற்நுட்பம், அனுகூலமான தீர்வைக் கொடுக்கும் முறையைப் பயன்படுத்தி ஒரு வேலையைக் காலமுறைப்படுத்துவதே இந்த ஆராய்ச்சியின் திட்டமாகும்.

'n' எண்ணிக்கை கொண்ட வேலைகளையும், 'm' எண்ணிக்கை கொண்ட இயந்திரங்களையும் பயன்படுத்தி ஒரு வேலையின் முடிவுறு நேரத்தைக் கண்டறிய வேண்டும். ஒரு அனுகூலமான முடிவுறு நேரத்தைக் கண்டறிய இடை விடாத வரிசைமுறைகளை உற்பத்தி செய்ய வேண்டும். மிகக் குறைவான முடிவுறு நேரத்தைக் கொண்ட வரிசைமுறையை அனுகூலமான தீர்வாக எடுத்துக் கொள்ள வேண்டும்.

ஜெனிடிக் வடிவமைப்பானது மிகச் சிறந்த வேகமான மற்றும் குவிவு தொழிற்நுட்பமாகும் . இதன் மூலம் எந்த எண்ணிக்கை கொண்ட வேலைகளையும், இயந்திரங்களையும் பயன்படுத்தி அனுகூலமான வரிசை முறையைக் கண்டறியலாம். மேலும் இந்த வடிவமைப்பு மற்ற வடிவமைப்புகளை விட மிகச் சிறந்த தீர்வினைக் கொடுக்கும்.

ஜெனிடிக் வடிவமைப்பு, இந்த ஆராய்ச்சிக்கு C என்ற காரியத்திட்டம் மூலம் செயல்படுத்தப்பட்டிருக்கிறது. மேலும் ஒரு சில நொடிகளில் எந்த எண்ணிக்கை கொண்ட வேலைகளையும் வரிசைப்படுத்திவிடலாம். இந்த ஆராய்ச்சித் திட்டத்தில் இருபதாவது வரிசைமுறை மிகக் குறைந்த முடிவுறு நேரத்தை அளிக்கிறது என்று கண்டறியப்பட்டுள்ளது இருபதாவது வரிசை முறையே அனுகூலமான தீர்வாகும்.

ACKNOWLEDGEMENT

The author would like to record her sincere thanks to her respectable guide **Dr.N.Gunasekaran**, Professor, Department of Mechanical Engineering,Kumaraguru College of Technology, Coimbatore for his motivation and guidance in completing the project work.

The author wishes to express her immense pleasure to **Dr.T.P.Mani**, Professor and Head of the Department, Department of Mechanical Engineering, Kumaraguru College of Technology, Coimbatore for his support and continuous encouragement not only in the completion of this project but also the course.

The author expresses her gratitude to her beloved Principal, **Dr.K.K.Padmanabhan**, Kumaraguru College of Technology, Coimbatore who has permitted to use the college facilities to complete the project work, in time.

The author finally would like to thank her beloved parents, friends and others who have helped her to reach this level of education.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

n	Number of jobs
m	Number of machines
t_j	Processing time
r_j	Ready time
d_j	Due date
C_j	Completion time
F_j	Flowtime
L_j	Lateness
T_j	Tardiness
F_{max}	Maximum Flowtime
T_{max}	Maximum Tardiness
N_t	Number of Tardy jobs
I/P	Input
O/P	Output
C_{max}	Completion time
F(X)	Objective function
P_c	Probability of Crossover
P_m	Probability of Mutation
X, Z, T	Connectors used in Flow chart
Y	Condition is satisfied
N	Condition is not satisfied
GA	Genetic Algorithm
FSPAC	Flow Shop with Availability Constraints
FSP	Flow Shop Problem

CHAPTER 1

INTRODUCTION TO SCHEDULING

1.1 INTRODUCTION

In our day-to-day life time is an important constraint. Every human being in this world plans or schedules his activities according to the time given to them. Planning activities and working according to it leads to a bright future. Engineering is a wide area where planning activities solve many problems such as more time consumption, delivery of products within due date, high penalty costs, scale of production etc. Planning activities and Scheduling are considered as the tracks of a train. Most industrial application deals with the scheduling problems like improper workflow balance, high makespan, increased flow time and lateness. Many heuristic / traditional techniques are available to solve the above said problems. The heuristic / traditional approaches consume more time when implemented for large number of unscheduled work or tasks. To rectify this problem there are many non-traditional techniques, which in combination with Information Technology produces optimum results. Moreover in heuristic procedures the objective function will be single i.e., makespan minimization or reduction in penalty cost or balancing workflow or reduction in idle time etc. Genetic algorithm is a technique which gives best results for multilevel objective function. Genetic Algorithm undergoes many iterations with many numbers of samples in each iteration and determines which schedule is the best for the task activity. "The Best way to predict Future is to create.. it."

1.2 SCHEDULING

Scheduling is the allocation of resources over time to perform a collection of tasks. It is a decision-making function: it is the process of determining a schedule. Scheduling is a body of theory: it is a collection of principles, models, conclusions that provide insight into the scheduling collection of tasks arises in a variety of situations. For example, in manufacturing applications the fundamental managerial questions involve selecting the product to be manufactured and scale of production. After market studies and economic analyses are used to resolve such issues, technological planning focuses on the question of how the product is manufactured. Only after these planning questions have been answered, and the availability of resources are known, it becomes appropriate to consider problems of scheduling.

1.3 SCHEDULING MODELS

The study of scheduling models contain the elements and relationships that frequently arise in scheduling problems, and they also suggest how feasible solutions are systematically be constructed. Format models are available to aid decision making in a variety of scheduling problems. For example, one of the simplest and most widely used models is the Gantt chart, which is a graphical representation of scheduling relationships. In its basic form the Gantt chart is a graph of resource allocation over time. Generally, specific resources are shown in vertical axis and a time scale is shown in horizontal axis. Figure 1.1 shows the Gantt chart foe scheduling 4 jobs on 3 resources.

In this concise format, analysis of the graphical relationships can yield inferences about the behavior of the given schedule, while manipulation of the graphical elements can yield comparative information about alternative scheduling decisions. In this way, the Gantt chart serves as a focus for implementing the systems approach in scheduling. The scheduling models are inherent in the systems approach and provide a direct decision making method when the systems approach is utilized. When a model is actually a faithful representation of reality, it can become an integral part of the scheduling function. It is the theory of scheduling that has developed this type of model, providing a useful framework for performing the scheduling function effectively.

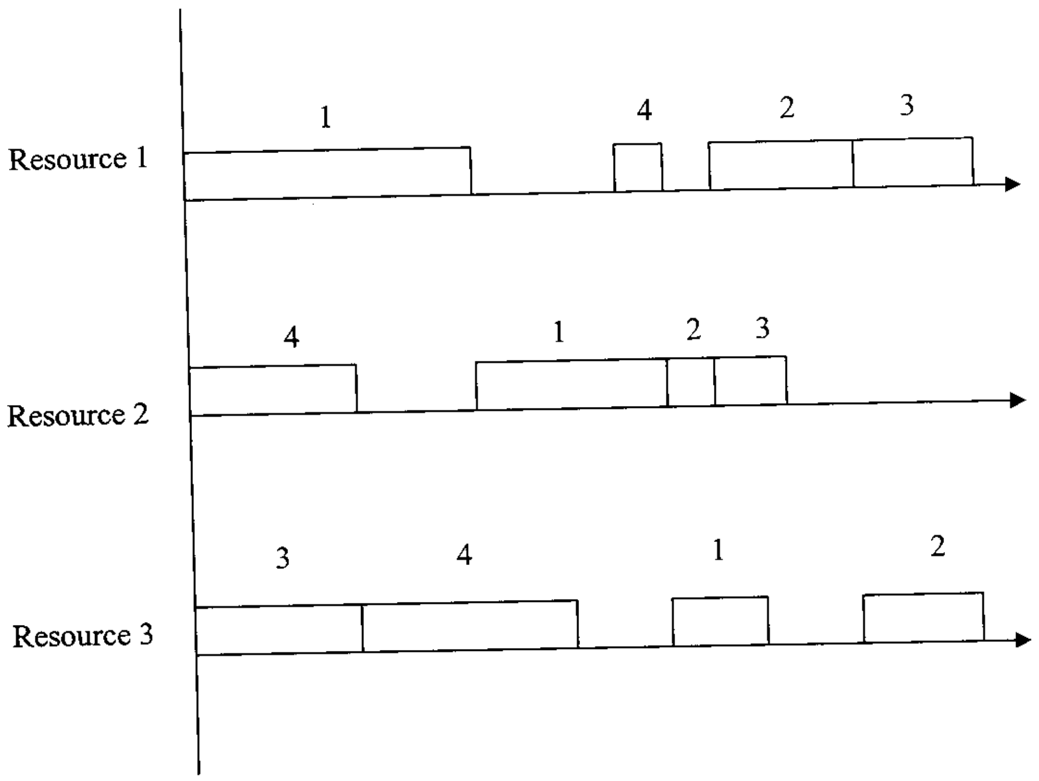


FIGURE 1.1 GANTT CHART FOR SCHEDULING 4 JOBS ON 3 RESOURCES.

In the above figure the resources used are called "machines", the basic task schedules are called "jobs". Sometimes, jobs may consist of some elementary tasks that are interrelated by precedence restrictions; such elementary tasks are referred to as "operations".

The pure sequencing problem is a specialized scheduling problem in which an ordering of the jobs completely determines a schedule. Moreover, the simplest pure sequencing problem is one in which there is a single resource, or machine. The basic single machine problem is characterized by these conditions.

i. A set of 'n' independent, single-operation jobs is available for processing at time zero.
ii. Setup times for the jobs are independent of job sequence and can be included in processing times.
iii. Job descriptors are known in advance.
iv. One machine is continuously available and is never kept idle while work is waiting,

Under these conditions, there is one-to-one correspondence between a sequence of the 'n' jobs and a permutation of the job indices 1, 2, …,n. The total number of distinct solutions to the basic single-machine problem is therefore n! which is the number of different permutations of 'n' elements.

1.4 TERMINOLOGIES USED

The following are the basic terminologies used in scheduling

i. *Processing time (t_j)*: The amount of processing required by job *j*.

ii. *Ready time (r_j)*: The point in time at which job *j* is available for processing.

iii. *Due date (d_j)*: The point in time at which the processing of job *j* is due to be completed.

Under the condition (ii) mentioned above, the processing time t_j will generally include both direct processing and facility setup time. The ready time can be thought of as an arrival time-the time when job j appears at the processing facility-and in the basic model the assumption in condition (i) is that $r_j = 0$ for all jobs. Due date may not be pertinent in certain problems, but meeting deadlines is a common concern, and the basic model can shed some light on deadline oriented objectives. Information that is generated as a result of scheduling decisions represents output from the scheduling function. In deterministic cases, scheduling decisions will determine the most fundamental piece of data to be used in evaluating schedules:

iv. *Completion time (C_j)*: The time at which the processing of job *j* is finished.

Quantitative measures for evaluating schedules are usually functions of job completion times. Two important quantities are:

v. *Flow time (F_j)*: The amount of time job j spends in the system:

$$F_j = C_j - r_j$$

vi. *Lateness (L_j)*: The amount of time by which the completion

These two quantities reflect two kinds of service. Flow time measures the response of the system to individual demands for service and represents the interval a job waits between its arrival and its departure. (This interval is sometimes called as turn around time). Lateness measures the conformity of the schedule to given due date and it is important to note that the lateness quantity takes on negative values whenever a job is completed early. Negative lateness represents the better service than requested, while positive lateness represents poorer service than requested. In many situations, distinct penalties and other costs will be associated with positive lateness, but no benefits will be associated with negative lateness. Therefore it is often helpful to work with a quantity that measures only positive lateness:

vii. *Tardiness (T_j):* The lateness of job *j* if it fails to meet its due date, or zero otherwise:

$$T_j = \max\ \{0, L_j\}$$

Schedules are generally evaluated by aggregate quantities that involve information about all jobs, resulting in one-dimensional performance measure. Measures of schedule performance are usually functions of the set of completion times in a schedule. For example, suppose that n jobs are to be scheduled. Aggregate performance measures that might be defined include the following:

i. Mean flow time: $\overline{F} = (1/n) \sum F_j$; where $j = 1, 2, \ldots, n$.
ii. Mean tardiness: $\overline{T} = (1/n) \sum T_j$; where $j = 1, 2, \ldots, n$.
iii. Maximum flow time: $F_{max} = \max\ \{F_j\}$; where $j = 1, 2, \ldots, n$.
iv. Maximum tardiness: $T_{max} = \max\ \{T_j\}$; where $j = 1, 2, \ldots, n$.
v. Number of tardy jobs: $N_t = \sum \delta\ (T_j)$; where $j = 1, 2, \ldots, n$.

1.5 CATEGORIES OF SCHEDULING

The main categories of scheduling are as follows

i. Job shop scheduling
ii. Flow shop scheduling

Job shop scheduling and flow shop scheduling differ from each other by the

Job shop scheduling

The classical job shop scheduling problem differs from flow shop problem in one important aspect: The flow of work (jobs) is not unidirectional. The elements of a problem are a set of machines and a collection of jobs to be scheduled. The most common formulation of the job shop problem specifies that each job have exactly *j* number of operations one on each machine *m*. Because the workflow in a job shop is not unidirectional, each machine in the shop can be characterized by the input and output flows of work as shown in Figure 1.2

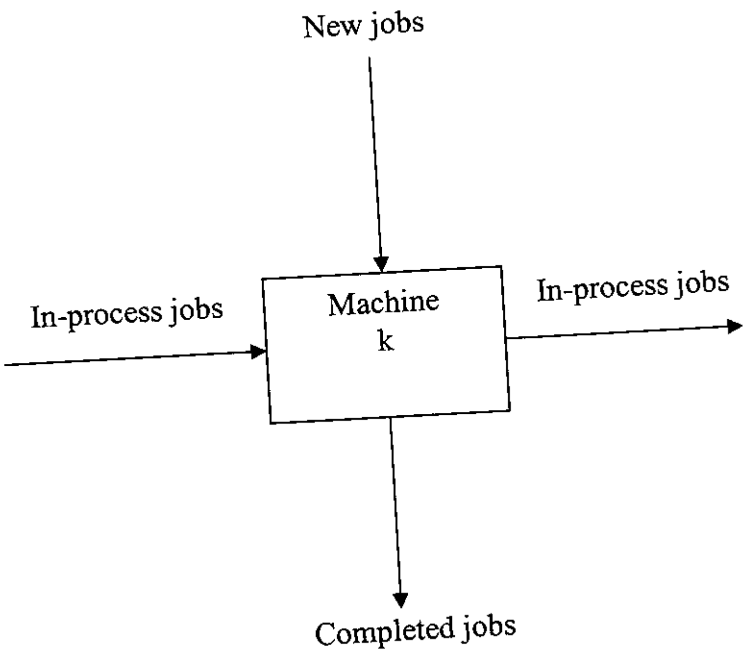


FIGURE 1.2 WORK FLOW AT A TYPICAL MACHINE IN JOB SHOP

Unlike the flow shop model, there is no initial machine that performs only the first operation of a job, nor there is a terminal machine that performs only the last operation of a job. In the job case it is more appropriate to describe an operation with a triplet *(i, j, k)* in order to denote the fact that operation 'j' of job 'i' requires machine 'k'. Let us now consider the graphical representation of a job shop problem by means of a Gantt chart. Figure 1.3 shows the job-by-job description of [illegible] It consists of a collection of blocks, each of which is

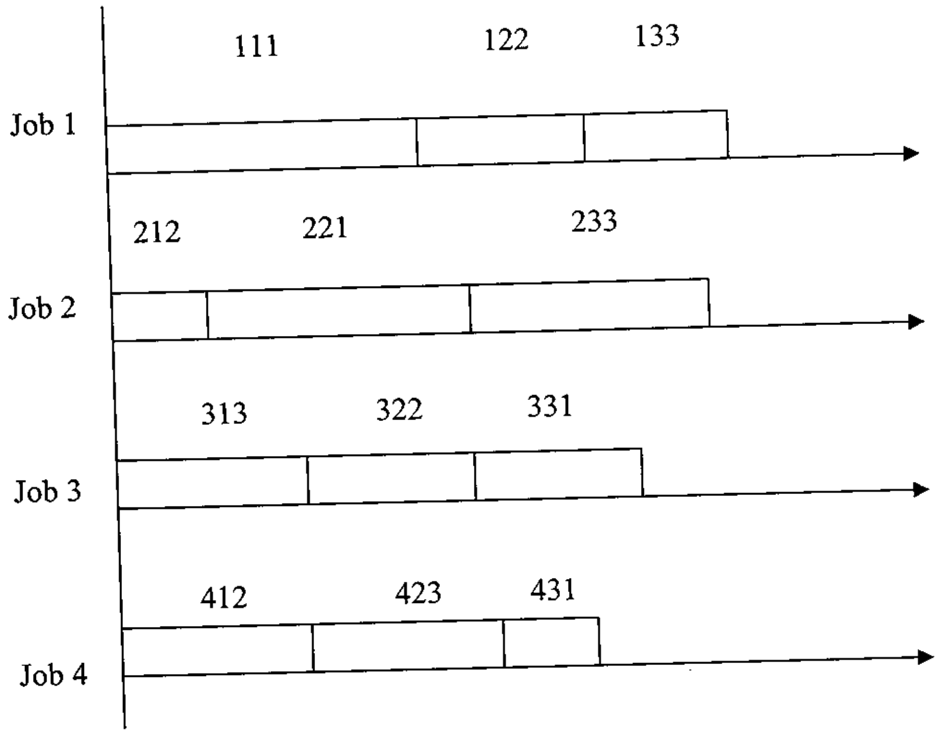
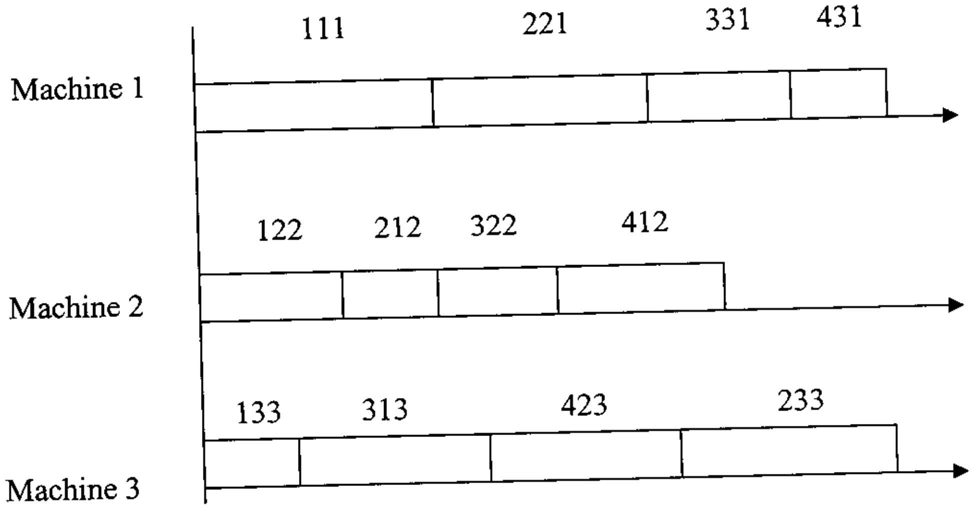


FIGURE 1.3 A JOB-BY-JOB DESCRIPTION OF A WORK.

One can also represent the schedule in a machine-by-machine description. In that Gantt chart machines are represented in vertical axis and the machine triplet sequence is shown along horizontal axis. Figure 1.4 explains the machine-by-machine description.



Though a work is scheduled according to machines or jobs there will be a feasible solution for any scheduling problem. A schedule is feasible solution of the logical constraints when all operations of each given job can be placed on one time axis in precedence order without overlap. A graphical display of this property is shown in Figure 1.5

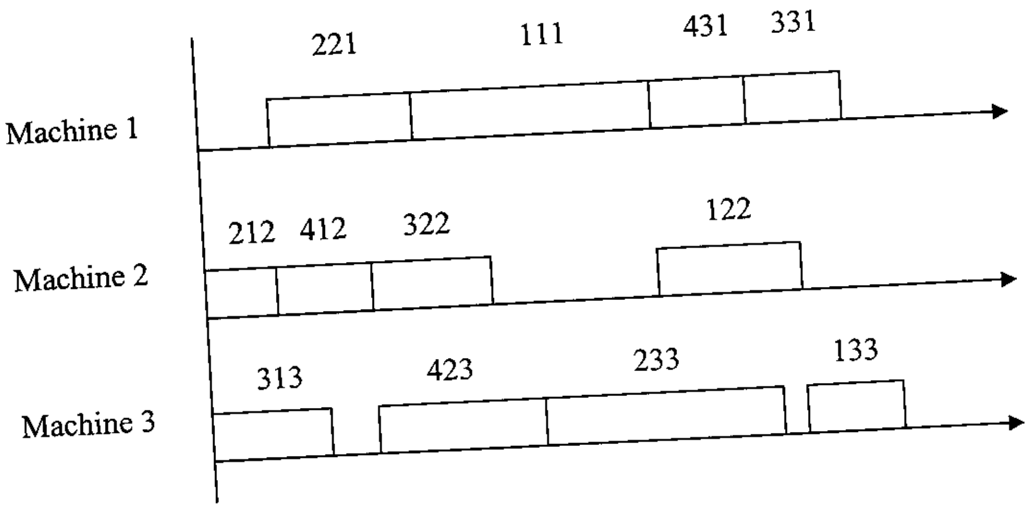


FIGURE 1.5 A FEASIBLE JOB SHOP SCHEDULE.

In the above figure the blank spaces represent the idle time. (2,3,3) represents 2nd job 3rd operation 3rd machine. The first problem in dealing with the job shop is to locate a similar "very good" subset of the feasible solutions for more detailed examination. This subset of schedules and how to generate these schedules systematically leads to the formation of a generation procedure. For these subset generation and decisions there are many dispatching rules. The Figure 1.6 represents the first level of classification schematically. Figure 1.7 displays the second level of classification by means of a schematic diagram.

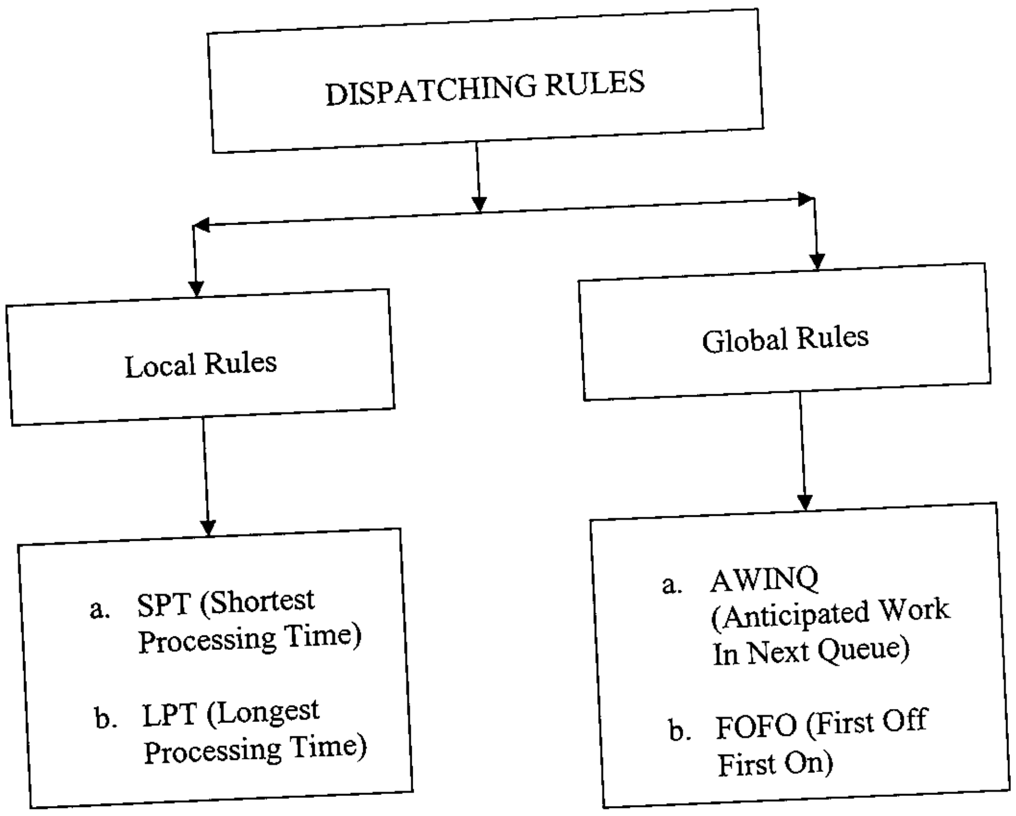

DISPATCHING RULES
Local Rules
Global Rules
a. SPT (Shortest Processing Time)
b. LPT (Longest Processing Time)
a. AWINQ (Anticipated Work In Next Queue)
b. FOFO (First Off First On)

FIGURE 1.6 SCHEMATIC REPRESENTATION OF CLASSIFICATION (FIRST LEVEL) OF DISPATCHING RULES.

In this way by using the local or global dispatching rules an optimum schedule is obtained for a job shop scheduling problem. But in flow shop scheduling problem these subsets, generation procedure etc. are not involved.

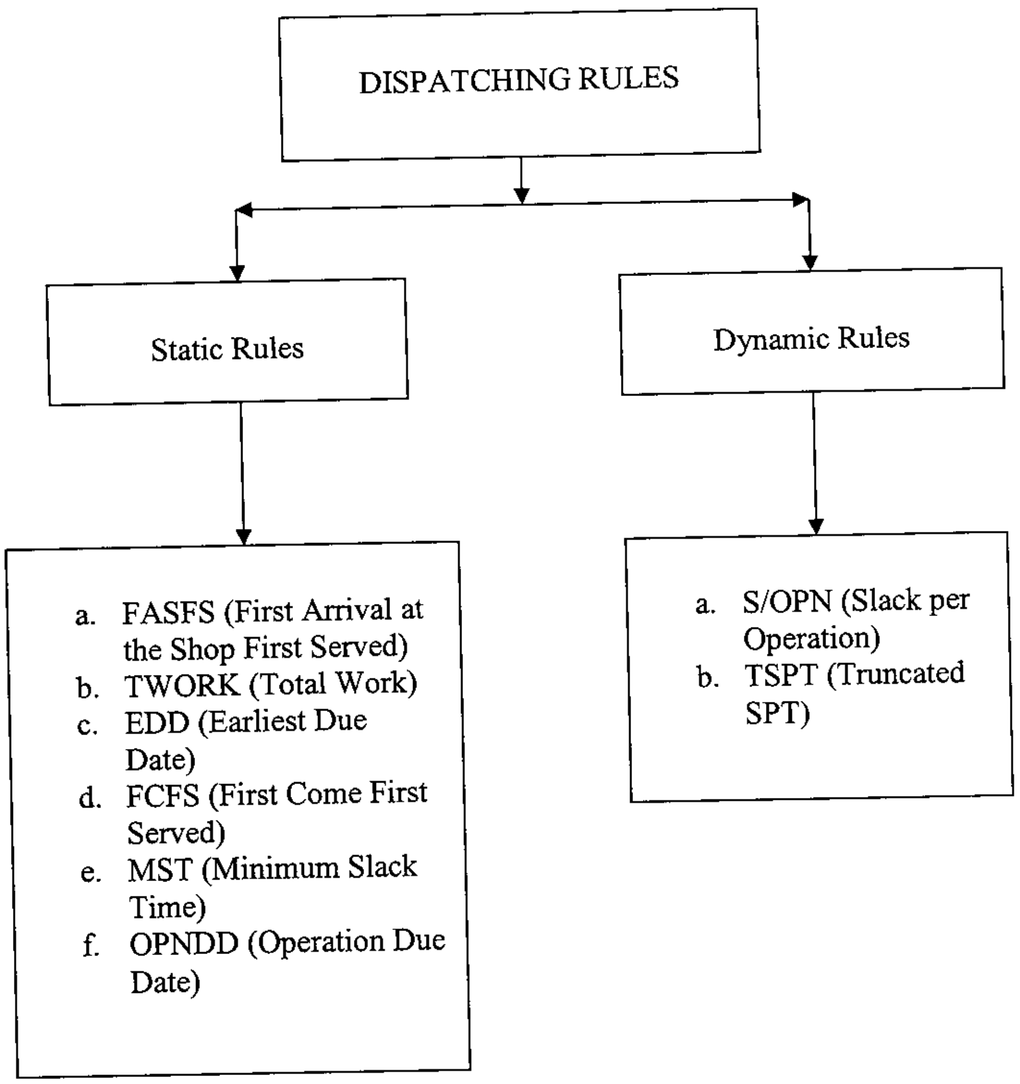

DISPATCHING RULES
Static Rules
Dynamic Rules
a. FASFS (First Arrival at the Shop First Served)
b. TWORK (Total Work)
c. EDD (Earliest Due Date)
d. FCFS (First Come First Served)
e. MST (Minimum Slack Time)
f. OPNDD (Operation Due Date)
a. S/OPN (Slack per Operation)
b. TSPT (Truncated SPT)

FIGURE 1.7 SCHEMATIC REPRESENTATION OF CLASSIFICATION (SECOND LEVEL) OFDISPATCHING RULES.

The following section explains how flow shop scheduling problem is being handled.

1.6 FLOWSHOP SCHEDULING

The flow shop is characterized by a flow of work that is unidirectional.

of work in a "pure" flow shop, in which all jobs require one operation on each machine. Figure 1.9 represents the flow of work in a more general shop. In this case, the jobs may require fewer than 'm' operations, that their operations may not always require adjacent machines in the numbered order, and that the initial and terminal operations may not always occur at machines 1 and 'm', respectively.

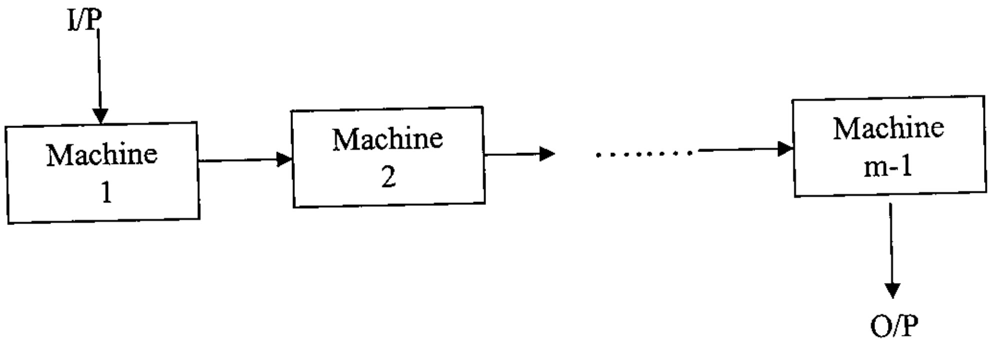


FIGURE 1.8 WORKFLOW IN A PURE FLOW SHOP.

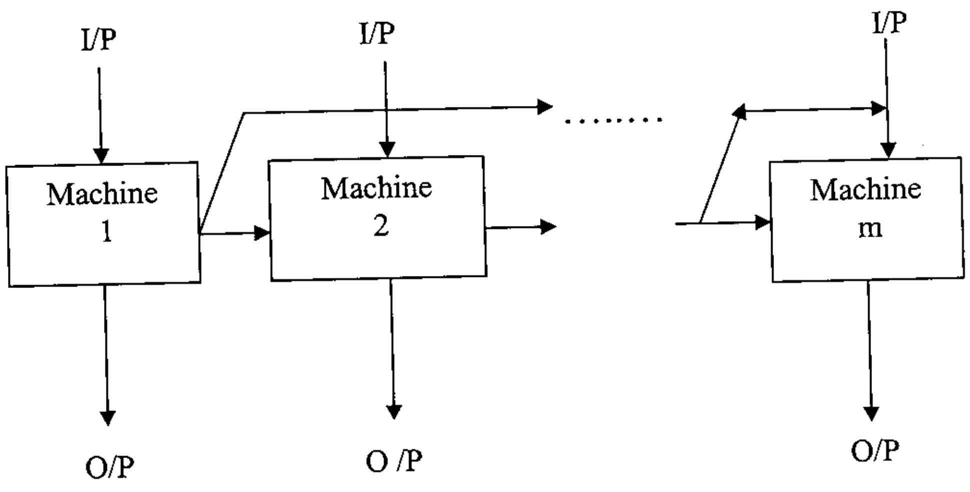


FIGURE 1.9 WORKFLOW IN A GENERAL FLOW SHOP.

The conditions that characterize flow shop problems are similar to the conditions of the basic single-machine model are,

i. A set of 'n' multiple-operation jobs is available for processing at time zero. (Each job requires 'm' operations and each operation requires a different machine).

ii. Set up times for the operations are sequence-independent and are included in processing times.

iii. Job descriptors are known in advance.

iv. 'm' different machines continuously available.

v. Individual operations are not preemptable.

In the flow shop case it may be necessary to provide for inserted idle time to achieve optimality. For example consider a two-job four-machine problem with the makespan of 14, shown in Figure 1.10. The schedule shown in Figure 1.11 is the optimum schedule since idle time is inserted at (1,3) before the second job has started its operation in machine 3.

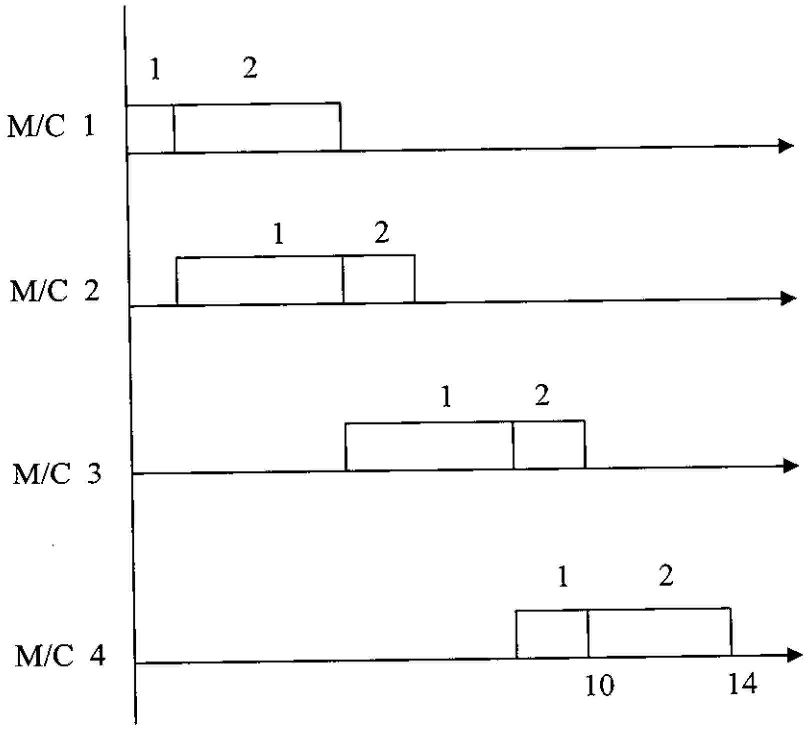


FIGURE 1.10 SCHEDULE WITH NO INSERTED IDLE

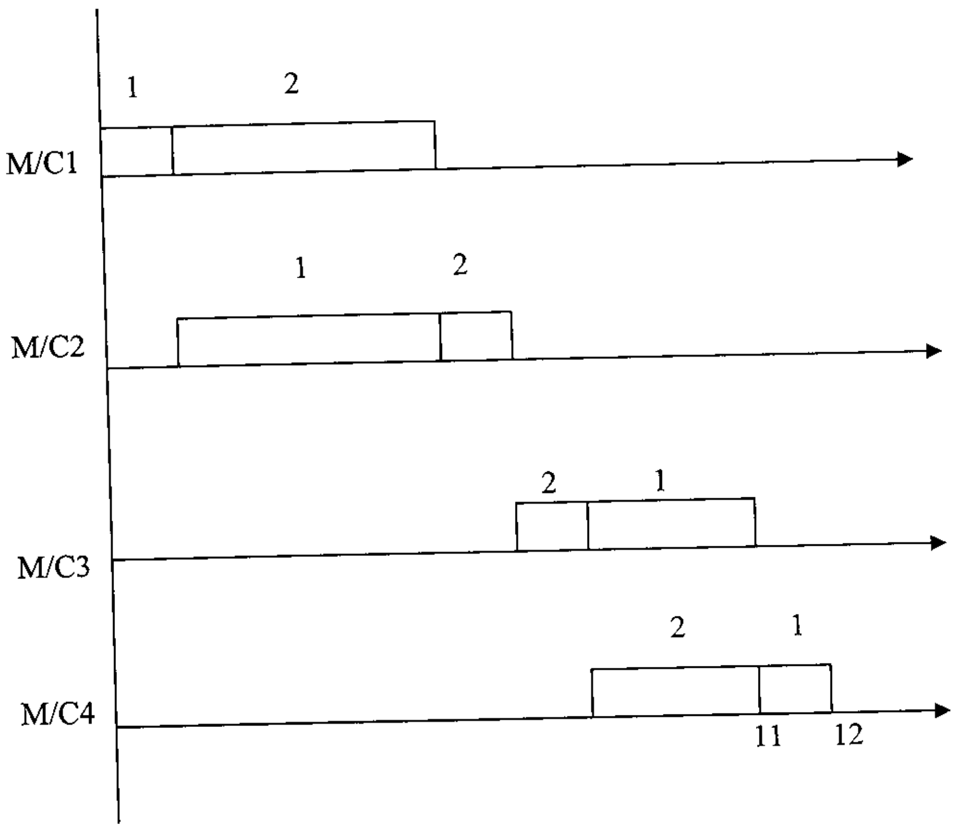


FIGURE 1.11 FLOW SHOP SCHEDULE WITH INSERTED IDLE TIME.

In flow shop problem the total number of distinct possible solutions for '*n*' jobs on '*m*' different machines can be examined by the permutation schedule (*n*!)^*m*. It is not always necessary to consider the above permutation schedule in determining an optimum solution. The two dominance properties given below indicate how much of a reduction is possible in flow shop problems.

Property 1: With respect to any regular measure of performance it is sufficient to consider only schedules in which the same job sequence occurs on machines 1 and 2.

Property 2: With respect to schedule makespan as a measure of performance it is sufficient to consider only schedules in which the same job sequence occurs on machines 1 and 2.

1.6 HEURISTIC METHODS

The heuristic algorithms/methods used to solve flow shop problems (some) are shown in Table 1.1

TABLE 1.1 HEURISTIC ALGORITHMS TO SOLVE FLOW SHOP PROBLEM.

AlGORITHMS/METHODS	NUMBER OF MACHINES	PURPOSE
i. Johnsons algorithm	2 or 3 machines	To reduce makespan
ii. Campbell Dudek& Smith algorithm	Multiple machines	-do-
iii. Palmer's heuristic method	Multiple machines	-do-
iv. Jackson's method	2 machines	-do-
v. Branch & Bound method	2 or more machines	-do-

Problem: A flow shop problem with 2 machines and 6 jobs is solved using Johnsons algorithm to be clear about flow shop scheduling.

Number of jobs $n = 6$

Number of machines $m = 2$.

The following table (Table 1.2) gives the processing times of each job performed on 2 machines.

TABLE 1.2 INFORMATION ABOUT A FLOW SHOP PROBLEM.

JOBS	MACHINE 1	MACHINE2
j	Processing Time	Processing Time
1	5	4
2	2	3
3	13	14
4	10	1
5	8	9
6	12	11

Procedure for Johnsons algorithm

Step 1: Find the minimum processing time (t_{i1},t_{i2}) from the Table 2.

t_{i1} = Processing time of job i on machine 1.

t_{i2} = Processing time of job i on machine 2.

Step 2: If the minimum processing time is listed under machine 1, place the associated job in the first available position of the sequence else

Step 3: If the minimum processing time is under machine 2, place the associated job in the last available position in the sequence.

Step 4: Remove the assigned job fro consideration and return to step 1 until all positions in the sequence are filled. Ties may be broken arbitrarily.

Solution

The schedule produced based on the above procedure is **2-5-3-6-1-4.**To calculate the makespan Gantt chart is used. Figure 1.8 shows how makespan is calculated.

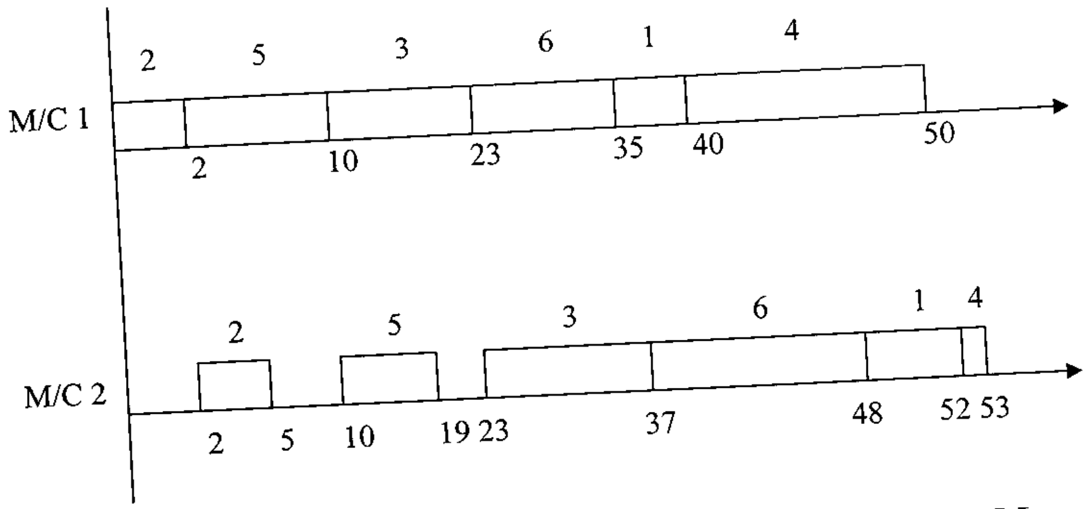


FIGURE 1.12 GANTT CHART FOR FLOW SHOP PROBLEM.

From the Figure 1.12 we can find the makespan as 53 time units. The following tables show the sequence formation (Table 1.3) and makespan calculation (Table 1.4). Table 1.3 describes how the jobs are scheduled from the unscheduled jobs and how it is positioned in the sequence. Table 1.4 describes the time taken for each job on each machine.

TABLE 1.3 SEQUENCE FORMATION FOR THE SCHEDULE 2-5-3-6-1-4.

STAGE	UNSCHEDULED JOBS	MINIMUM T(j,m)	ASSIGNMENT	PARTIAL SEQUENCE
1	1,2,3,4,5,6	T (4,2)	4[6]	_ _ _ _ _ 4
2	1,2,3,5,6	T (2,1)	2[1]	2 _ _ _ _ 4
3	1,3,5,6	T (1,2)	1[5]	2 _ _ _ 1 4
4	3,5,6	T (5,1)	5[2]	2 5 _ _ 1 4
5	3,6	T (6,2)	6[4]	2 5 _ 6 1 4
6	3	T (3,1)	3[3]	2 5 3 6 1 4

TABLE 1.4 CALCULATION OF MAKESPAN.

JOBS	MACHINE 1 (Time In)	MACHINE 1 (Time Out)	MACHINE 2 (Time In)	MACHINE 2 (Time Out)
2	0	2	2	5
5	2	10	10	19
3	10	23	23	37
1	23	35	37	48
6	35	40	48	52
4	40	50	52	**53**

1.7 REASONS FOR GOING TO NON-TRADITIONAL TECHNIQUES

From the above described problem it will be easy to understand how to solve a flow shop problem with *n* number if jobs and *m* number of machines. We can calculate makespan using Gantt chart as shown in Figure 1.11 or through the Table 4. When we have to solve a problem with more number of jobs (say more than 20) and machines (say more than 6 or 7), then we should go for optimization techniques.

In the development of scheduling models more general than single-machine models, the flow shop represents the most direct extension to multiple-resource situations. In general, solutions to flow shop problem appear to require either combinatorial techniques that take advantage of special problem structures or else clever heuristic methods. The effectiveness of any solution method can be evaluated only on relative terms, on the basis of the availability of computational resources and the acceptability of sub optimal results. From the above statements we can say or conclude that flow shop requires some computational techniques, which should be capable of giving optimal results. Hence, we go for Genetic

CHAPTER 2

LITERATURE SURVEY

Scheduling problems are found to emerge as new situations and constraints in the new life manufacturing environment. Researchers are attempting such scheduling problems and trying to minimum time. A literature survey covering the fkow shop scheduling, makespan minimization and Genetic Algorithm is presented in this section.

Riad Aggoune (2004) deals with the scheduling of a flow shop with availability constraints (FSPAC). In such a problem, machines are not continuously available for processing jobs due to a preventive maintenance activity. In this paper, two variants of the non-preemptive FSPAC are considered. In the first variant, starting times of maintenance tasks are fixed while in the second one the maintenance tasks must be performed on given time windows. Since the FSPAC is NP-hard in the strong sense, a heuristic approach based on a genetic algorithm and a tabu search is proposed to approximately solve the makespan minimization problem. Computational experiments are performed on randomly generated instances to show the efficiency of the proposed approaches.

Rubén Ruiz et al., (2006) proposes new genetic algorithms for solving the permutation FSP that prove to be competitive when compared to many other well known algorithms. The optimization criterion considered is the minimization of the total completion time or makespan (C_{max}). A robust genetic algorithm and a fast hybrid implementation is shown in this paper. These algorithms use new genetic operators, advanced techniques like hybridization with local search and an efficient population initialization as well as a new generational scheme. A complete evaluation of the different parameters and operators of the algorithms by means of a Design of Experiments approach is also given. The algorithm's effectiveness is compared against 11 other methods, including genetic algorithms, tabu search, simulated annealing and other advanced and recent techniques.

Iyer, S. K., et al., (2004), in their paper have shown that the efficiency of GAs in solving a flowshop problem can be improved significantly by tailoring the various GA operators to suit the structure of the problem. The flowshop problem is one of scheduling jobs in an assembly line with the objective of minimizing the completion time or makespan. They compare the performance of GA using the standard implementation and a modified search strategy that tries to use problem specific information. They present empirical evidence via extensive simulation studies supported by statistical tests of improvement in efficiency.

Allahverdi, A., (2003), in his paper considers the flowshop scheduling problem with the objective of minimizing the weighted sum of makespan and mean flowtime. In this paper, a comparison of already available heuristics in the literature is conducted. Moreover, three new heuristic algorithms are proposed, which can be utilized for both the two-machine and *m*-machine problems. Computational experiments indicate that the proposed heuristics are superior to all the existing heuristics in the literature including a genetic algorithm. Two dominance relations are also developed; one for the two-machine and the other for the three-machine problem. Experimental results show that both relations are efficient.

Neppalli, V. R., et al., (1996), considers the two-stage bicriteria flow shop scheduling problem with the objective of minimizing the total flow time subject to obtaining the optimal makespan. Two Genetic Algorithms (GA) based approaches are proposed to solve the problem. The effectiveness of the proposed GA based approaches is demonstrated by comparing their performance with the only known heuristic for the problem. The computational experiments show that the proposed GA based approaches are effective in solving the problem and recommend that the proposed GA based approaches are useful for solving the multi-machine, multi-criteria scheduling problems.

Portmann M. C., et al., (1998), article deals with an optimal methods for solving a *k*-stage hybrid flowshop scheduling problem. The aim of this article is to present an improvement of Branch and Bound algorithm. The author introduces the upper bound. More precisely, GA takes into account the set of partial decisions made by the branch and bound and builds a series of populations of complete solutions with the aim of improving the upper bound (the best found criterion value corresponding to a complete solution).

Montanna D., et al., (2000) deals with real time scheduling of large-scale problems such as large and complex search spaces, dynamically changing problems and a variety of problem dependent constraints and preferences. Genetic algorithm is used due to their adaptability and effectiveness at searching large spaces. Domain-specific chromosome representation, multi-objective evaluation function, dynamic rescheduling and cooperative interaction with human operators are the key aspects discussed in this paper.

Rajakumar, S., et al., (2004), explains how workflow is balanced in Parallel machine scheduling. Work flow is the total time during which the work centres are busy. Idle time is not taken into account when calculating workflow. In parallel machine scheduling 'm' machines to which 'n' jobs are to be assigned based on different priority strategies such as RANDOM, LPT (Longest Processing Time), SPT (Shortest Processing Time). Relative percentage of imbalance (RPI) is adopted among parallel machines to evaluate the performance of these strategies in a standard manufacturing environment. The LPT rule shows better performance for the combination of larger job sizes and higher number of workcentres or machines. A computer program was coded in C++ language for validation.

Ravichandran, K. S., et al., (2002), deals with the optimization of Flexible Manufacturing Systems (FMS) using Fuzzy and Genetic Algorithm techniques.. Decision making process in a manufacturing system involves uncertainties and ambiguities. Fuzzy logic and Genetic algorithm solves these uncertainties and ambiguities. Part family formation is dealt in this paper. A new coefficient measure was developed and was used to form a part family. A mathematical model that uses this similarity coefficient for solving the part-family formation problems optimally in an FMS was developed. A balanced workload and an [illegible] is obtained by fuzzy and Genetic algorithm.

Gopalakrishnan, A., et al, (2005), deals with the optimization of job shop scheduling problem using non-traditional optimization techniques such as Tabu search, Genetic Algorithm and Simulated Annealing. For a given number of jobs and machines an optimum schedule is obtained. The schedule obtained through these techniques are compared by framing these techniques as a C program.. All the three methods produces optimum results. The advantage is that the result is obtained within minutes while running on a standard PC.

CHAPTER 3

PROBLEM DEFINITION

FLOW SHOP PROBLEM DEFINITION

A general flow shop containing *n* number of jobs and *m* number of machines is taken to find the optimum makespan using Genetic algorithm. The jobs perform some operations on the machines to obtain a product. As already explained in a flow shop the job entering the first machine leaves the terminal machine after processing in all the machines. Also the processing time varies for each job while performing in different machines.

Here, Number of jobs, 'n'=10

Number of machines, 'm'=2.

The following table displays the information about the problem taken. The processing time and batch quantity are given in the Table 3.1

TABLE 3.1 INFORMATION ABOUT THE PROBLEM

JOBS	PROCESSING TIME OF MACHINE 1	PROCESSING TIME OF MACHINE 2	BATCH QUANTITY
0	10	10	200
1	30	20	100
2	10	10	100
3	20	10	100
4	10	20	150
5	10	10	200
6	10	10	150
7	10	20	125
8	30	30	100

Source: Ravichandran, K.S., Chandra Sekhara Rao, K., Saravanan, R., (2002), The Role of Fuzzy and Genetic Algorithms in Part Family Formation and Sequence Optimization for Flexible Manufacturing Systems, *International Journal On Manufacturing Technology.*

Processing time is the time taken by a job to perform its operation in a given machine for a single quantity. Batch quantity is the amount of products to be produced for the given job.

CHAPTER 4

GENETIC ALGORITHM

4.1 FUNDAMENTALS OF GENETIC ALGORITHM (GA)

Decision features occur in all fields of human activities such as scientific and technological and affect every sphere of our life. Engineering design, which entails sizing, dimensioning, and detailed element planning is also not exempt from its influence. For example an aircraft wing can be made from aluminum or steel and once material and shape are chosen, there are many methods of devising the required internal structure. In civil engineering also, designing a roof to cover large area devoid of intermediate columns requires optimal designing.

The aim to make objective function a maximum or minimum, that is, it is required to find an element X1 in A if it exists such that

$$F(X_1) <= F(X), \text{ for minimization}$$

$$F(X) <= F(X_1), \text{ for maximization}$$

The following major questions may arise in this process

i. Does an optimal solution exist?

ii. Is it unique?

iii. What is the procedure?

iv. How sensitive the optimal solution is?

v. How the solution behaves for small changes in parameters?

Since 1040, several optimization problems have not been tackled by classical procedures including:

1. Linear programming
2. Transportation
3. Assignment
4. Nonlinear programming
5. Dynamic programming
6. Inventory
7. Queuing

Nontraditional search and optimization methods have become popular in engineering optimization problems in recent past. These algorithms include:

i. Simulated annealing
ii. Ant colony optimization
iii. Random cost
iv. Evolution strategy
v. Genetic algorithm
vi. Cellular automata.

4.1.1 Basic concepts

Genetic algorithms are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solutions which we might not find in a life time.

GA is very different from most of the traditional optimization methods. GA need design space to be converted into genetic space. So, genetic algorithms work with a coding of variables. The advantage of working with a coding of variable space is that coding discretizes the search space even though the function may be continuous. A more striking difference between genetic algorithms and most of the traditional optimization methods is that GA uses a population of points at one time in contrast to the single point approach by traditional optimization methods. This means that GA processes a number of designs at the same time. Traditional optimization methods uses transition rules, they are deterministic in nature where GA uses randomized operators. Random operators improve the search space in an adaptive manner. Three most objectives of using GA are:

i. Definition of objective function.
ii. Definition and implementation of genetic representation.
iii. Definition and implementation of genetic operators.

Once these three have been defined, the GA should work fairly well beyond doubt. We can, by different variations, improve the performance, find multiple optima or parallelize the algorithms.

4.1.2 Working principle

To illustrate the working principle of GA, let us consider the following maximization problem.

Maximize $f(X)$

X_i (lower) <= X <= X_i (upper) for i = 1,2,.., N

If we want to minimize f(X), for f(X) > 0, then we can write the objective function as

Maximize = 1/(1+f(X))

If f(X) < 0 instead of minimizing f(X), maximize {-f(X)}. Hence, both maximization and minimization problems can be handled by GA.

4.1.3 Encoding

As already said the GA variables are coded. There are many ways of representing individual genes; they are bits, arrays, trees, lists or any other object. The following section deals with the types of encoding.

i. Binary encoding.
ii. Octal encoding.
iii. Hexadecimal encoding.
iv. Permutation encoding.
v. Value encoding.
vi. Tree encoding.

This project work deals with permutation encoding since the job sequence varies for every schedule in each iteration. Simply by using the numbers, here chromosomes directly we can easily identify the sequence.

4.1.4 Fitness function

GAs mimics the Darwinian theory of survival of the fittest and principle of [illegible] process. Therefore, GAs are usually suitable for solving maximization problems by some suitable transformation. In general, fitness function F(X) is first derived from the objective function and used in successive genetic operations. Certain genetic operators require that fitness function be non-negative, although certain operators do not have this requirement. Consider the following transformations

$$F(X) = f(X) \text{ for maximization problem}$$

$$F(X) = 1/(f(X)) \text{ for minimization problem, if } f(X) \neq 0$$

$$F(X) = 1/(1+f(X)), \text{ if } f(X) = 0.$$

A number of such transformations are possible. The fitness function value of the string is known as *string's fitness.*

4.2 OPERATORS OF GA

A simple Genetic algorithm largely uses the three main basic operators, they are

i. Reproduction
ii. Crossover
iii. Mutation

4.2.1 Reproduction

Reproduction is the first operator applied on population. Chromosomes are selected from the population to be parents to crossover and produce offspring. According to Darwin's evolution theory of survival of the fittest, the best ones should survive and create offspring. For this reason reproduction operator is sometimes known as the selection operator. There exists a number of reproduction operators in GA literature but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in the probabilistic manner. The various methods of selecting chromosomes for parents to crossover are:

i. Roulette-wheel selection

iv. Rank selection

v. Steady-state selection

Rank selection operator is used in this project work. This operator first ranks the population and takes every chromosome, receives fitness from the ranking. The worst will have the fitness 1, the next 2… And the best will have fitness N (N is the number of chromosomes I the population). Figure 4.1 shows how a Roulette wheel is designed according to rank.

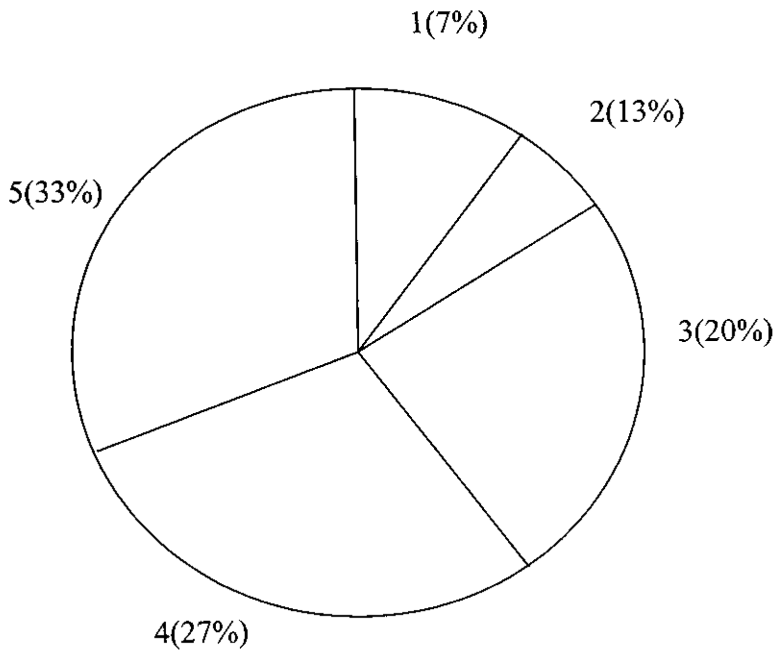


FIGURE 4.1 ROULETTE WHEEL ACCORDING TO RANK.

4.2.2 Crossover

After the reproduction phase is over, the population is enriched with better individuals. Reproduction makes clones of good strings, but does noe create new ones. Crossover operator is applied to the mating pool with the hope it would create a better string. The aim of the crossover pointer is to search the parameter space. In addition search is made in a way such that the information stored in the present string is maximally preserved because these parent strings are instances of good strings selected during reproduction. Crossover is a recombination operator, which proceeds in three steps. First the reproduction operator selects at random a following the cross-site. There exists many types of crossover operations in genetic algorithm which are as follows:

i. Single-site crossover
ii. Two-point crossover
iii. Multi-point crossover
iv. Uniform crossover
v. Matrix crossover (2D crossover)

Single-site crossover has been used in this project work. In this type of crossover, a cross site ia selected randomly along the length of the mated strings and bits next to the cross-sites are exchanged as shown below. If appropriate site is chosen, better children can be obtained by combining good substances of parents. Hence better children are produced and they will continue in the next generation also. But if good strings are not crested by crossover, they will not survive beyond next generation because reproduction will not select those strings for the next mating pool.

Crossover rate

In GA literature, the term crossover rate is usually denoted as P_c, the probability of crossover. The probability varies from 0 to1. This is calculated in genetic algorithm by finding out the ratio of the number of pairs to be crossed to some fixed population. Typically for a population size of 30 to 200, crossover rates are ranged from 0.5 to 1. With random sites, the children strings produced may not have a combination of good substrings from parent strings depending on whether or not the crossing site falls in the appropriate place. If good strings are not created by crossover, they will not survive too long, because reproduction will select against those strings in subsequent generations. It is clear from this discussion that the effect of crossover may either be detrimental or beneficial.

For example, let us consider the following scheduling sequence, if the crossover-site is 6,

Parent - 1 9 4 3 7 1 5 | 2 0 6 8

Child – 1 9 4 3 7 1 5 | 2 8 0 6

Child – 2 2 7 5 4 8 0 | 9 3 1 6

String after mating

4.2.3 Mutation

After crossover, the strings are subjected to mutation. Mutation of a bit involves flipping it, changing it 0 to 1 and vice versa with a small mutation probability P_m. The bit-wise mutation is performed bit-by-bit by flipping a coin with a probability of P_m. Flipping a coin with a probability is simulated as follows. A number between 10 to1 is chosen at random. If the random number is smaller than Pm then the outcome of coin flipping is true, otherwise the outcome is false. If at any bit, the outcome is true then the bit is altered, otherwise the bit is kept unchanged. The bits of the strings are independently muted, that is, the mutation of a bit does no affect the probability of mutation of other bits.

Mutation rate

Mutation rate is the probability of mutation, which is used to calculate number of bits to be muted. The mutation operator preserves the diversity among the population, which is also very important for the search. Mutation probabilities are smaller in natural populations leading us to conclude that mutation is appropriately considered a secondary mechanism of genetic algorithm adoption. Typically the simple genetic algorithm uses the population size of 30 to 200 with the mutation rates varying from 0.001 to 0.5. Consider the following schedule with the mutation operator at site (3,7) then the resulting schedule will be,

Before mutation - 9 4 **3** 7 1 5 **2** 8 0 6

After mutation - 9 4 **2** 7 1 5 **3** 8 0 6

4.3 ADVANTAGES OF GA OVER TRADITIONAL METHODS

As seen from the working principles of GA, GAs are radically different with a coding of variable are that coding discretizes the search space even though the function may be continuous. On the other hand, since GA requires only function values at discrete points, a discrete or continuous function can be handled with no extra cost. This allows GA to be applied to a number of problems. Genetic algorithm exploits the similarities in string structure to make an effective search. Genetic algorithm works with the population of points instead of a single point. In GA, previously found good information is emphasized using reproduction operator and propagated adaptively through crossover and mutation operators. Genetic algorithm is a population based search algorithm and multiple optimal solutions can be captured, thereby reducing the effort to use the algorithm many times. Summarizing the concepts GA has the following benefits,

i. Easy to understand,
ii. Modular, separate from application,
iii. Supports multi-objective optimization,
iv. Good for noisy environment,
v. We always get an answer and the answer gets better with time,
vi. Inherently parallel and easily distributed,
vii. There are many ways to speed up and improve a GA's basic applications as knowledge about the problem domain is general,
viii. Easy to exploit for previous or alternate solutions,
ix. Flexible in forming building blocks for hybrid applications, and
x. Has substantial history and range of use.

Genetic Algorithm is used where

i. alternate solutions are too slowly or overly complicated
ii. need an exploratory tool to examine new approaches
iii. problem that is similar to one that has already been successfully soved by GA
iv. we want to hybridize existing solutions
v. benefits of GA technology meet key problem requirements

The applications of GA are shown in Table 4.1.

TABLE 4.1 GA APPLICATIONS

DOMAINS	APPLICATION TYPES
Control	GAs pipeline, pole balancing, missile evasion, pursuit
Design	Semi conductor layout, aircraft design, keyboard configuration, communication network
Scheduling	Manufacturing, facility scheduling, resource allocation
Robotics	Trajectory planning
Machine learning	Designing neural networks, classification algorithms
Signal processing	Filter design
Game playing	Poker, checker, prisoner's dilemma
Combinatorial optimization	Set covering, traveling salesman, routing, bin packing, graph coloring and partitioning

CHAPTER 5

ALGORITHM AND FLOW CHART

This section deals how Genetic algorithm is used to solve the flow shop scheduling problem.

5.1 STEP BY STEP PROCEDURE OF GA

Step 1: Generate 30 random schedule with 'n' number of jobs and 'm' number of machines.

Example: n = 10, m = 7.

The schedules generated using random number generator will be,

1) 4-5-3-8-9-0-6-2-7-1
2) 3-9-5-1-7-0-5-4-7-6
3) ....

.

.

30) 9-6-0-2-3-7-8-1-5-4

Step 2: Reproduction - Calculate the makespan for each schedule generated above and save the minimum makespan.

Step 3: Rank selection - Rank the makespan i.e. highest makespan as rank number one and the one with the lowest as last rank

Step 4: Arrange the schedule sequence in descending order according to the rank selected.

Step 5: Crossover - Generate 15 random numbers for each pair of schedules.

Step 6: Determine the probability of cross over (P_c) with 0.1 unit of accuracy (as explained in 4.2.2). If the generated random number is less than P_c accept the schedule for crossover else reject the schedule i.e. it does not undergoes crossover operation.

Step 7: Perform crossover operation (as explained in 4.2.2).

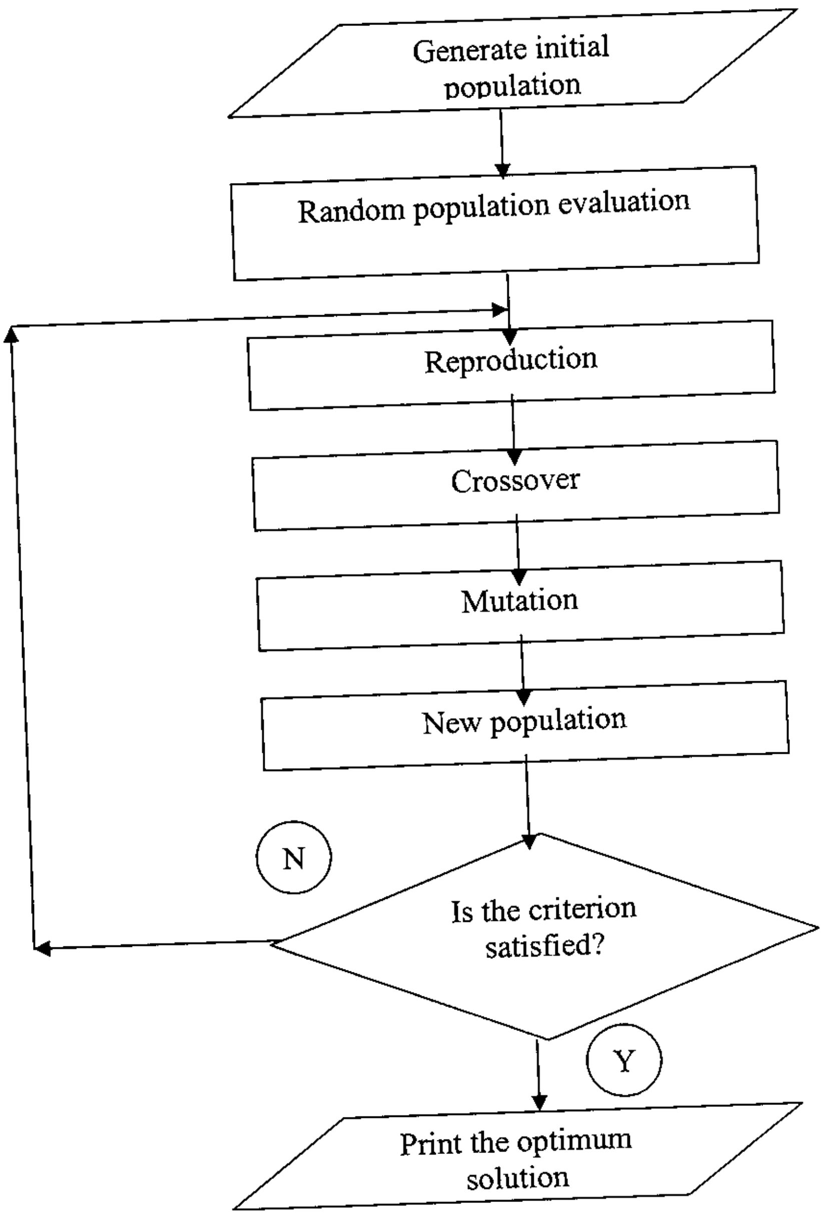
Step 8: Mutation – Generate 30 random numbers by random number generator for

Step 9: Determine the probability of mutation (P_m) with 0.01 unit of accuracy (as explained in 4.2.2). If the random number generated is less than P_c accept the schedule to mutate else reject i.e. it fails to undergo mutation operation.

Step 10: Perform mutation operation (as explained in 4.2.3).

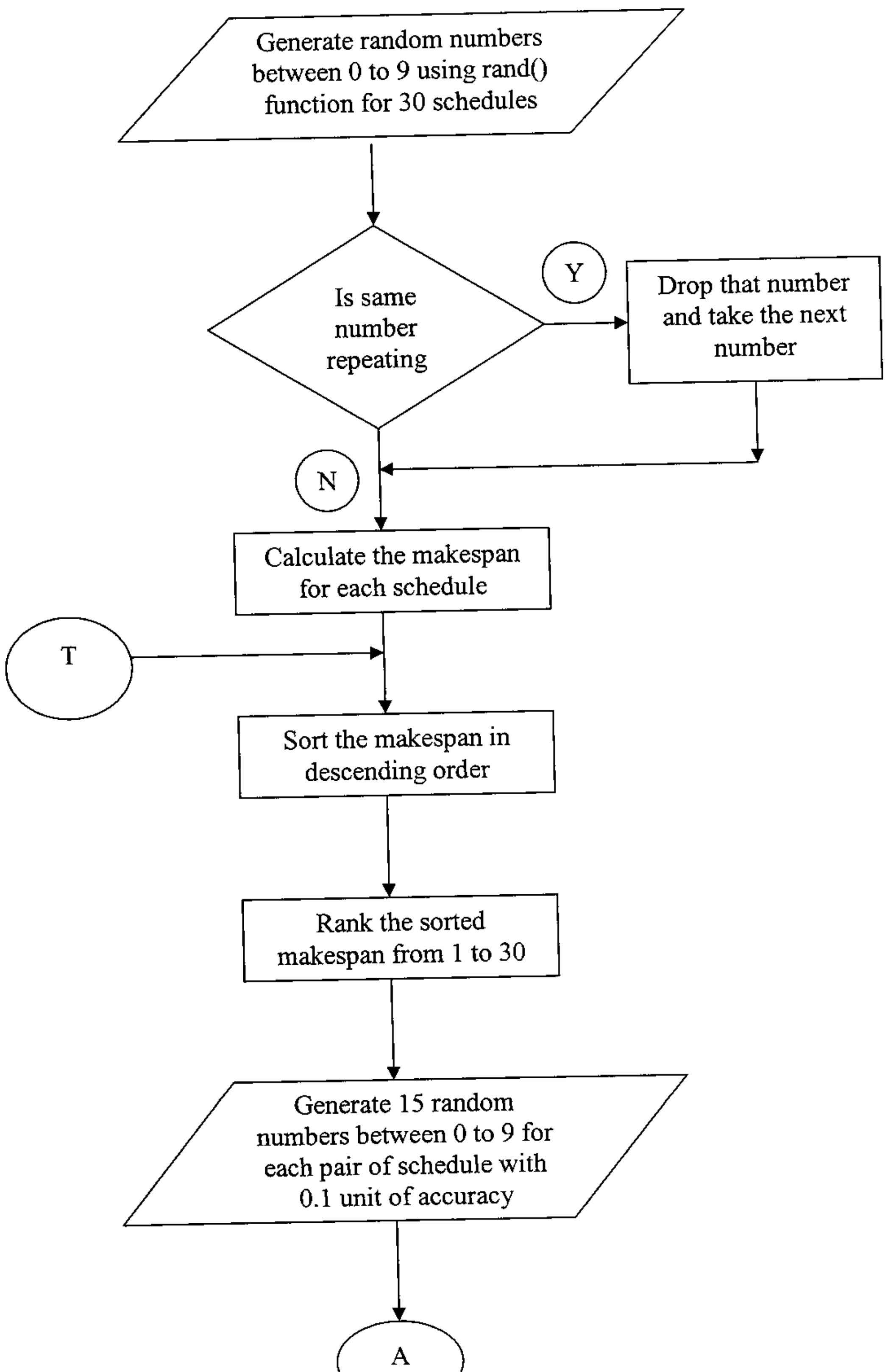
Step 11: Now calculate the makespan for the newly generated scheduling sequences (for 30 samples) and save the minimum makespan. (End of first iteration)

Step 12: Repeat the steps from step 3 until a same minimum makespan is saved.

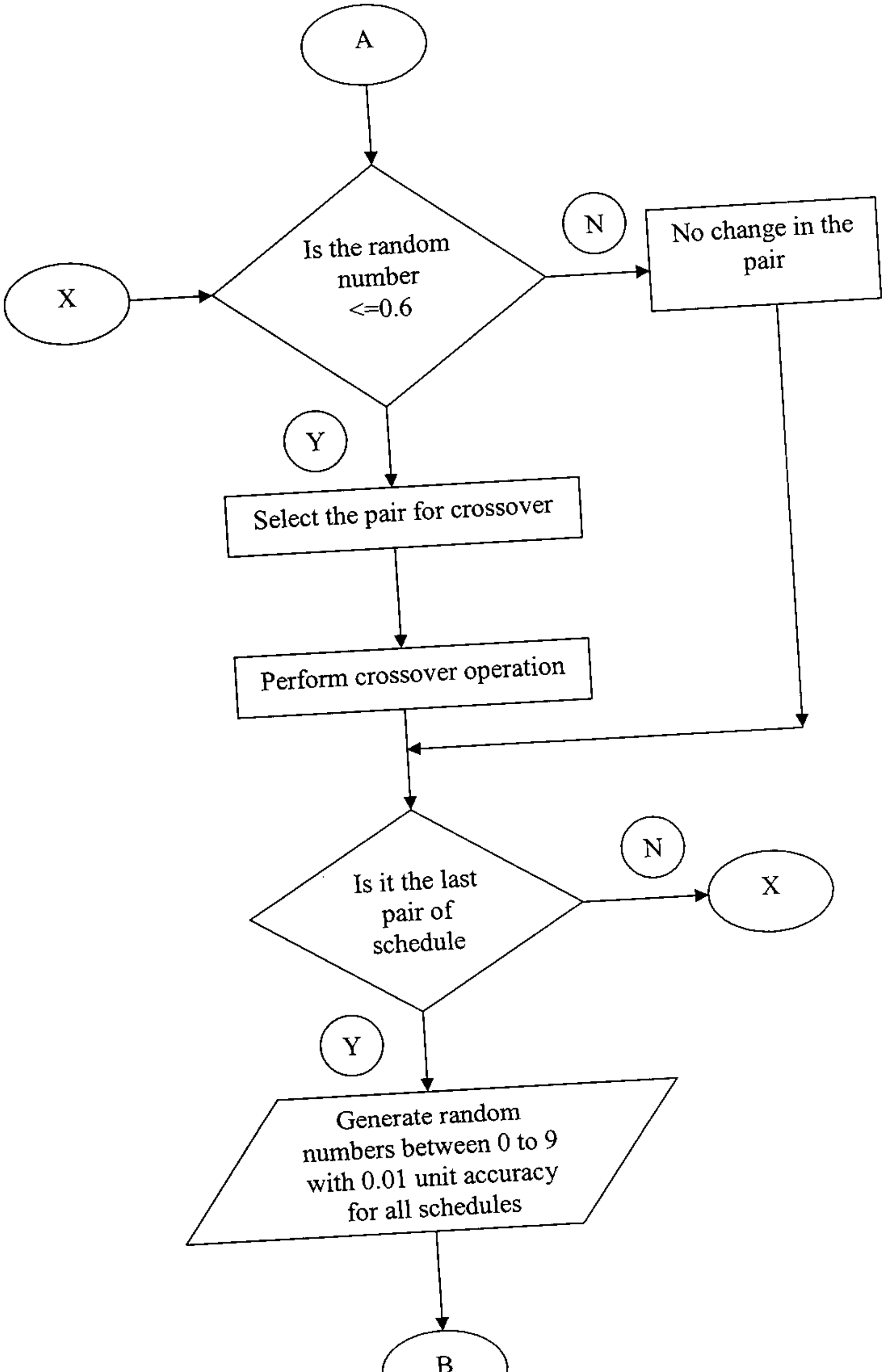


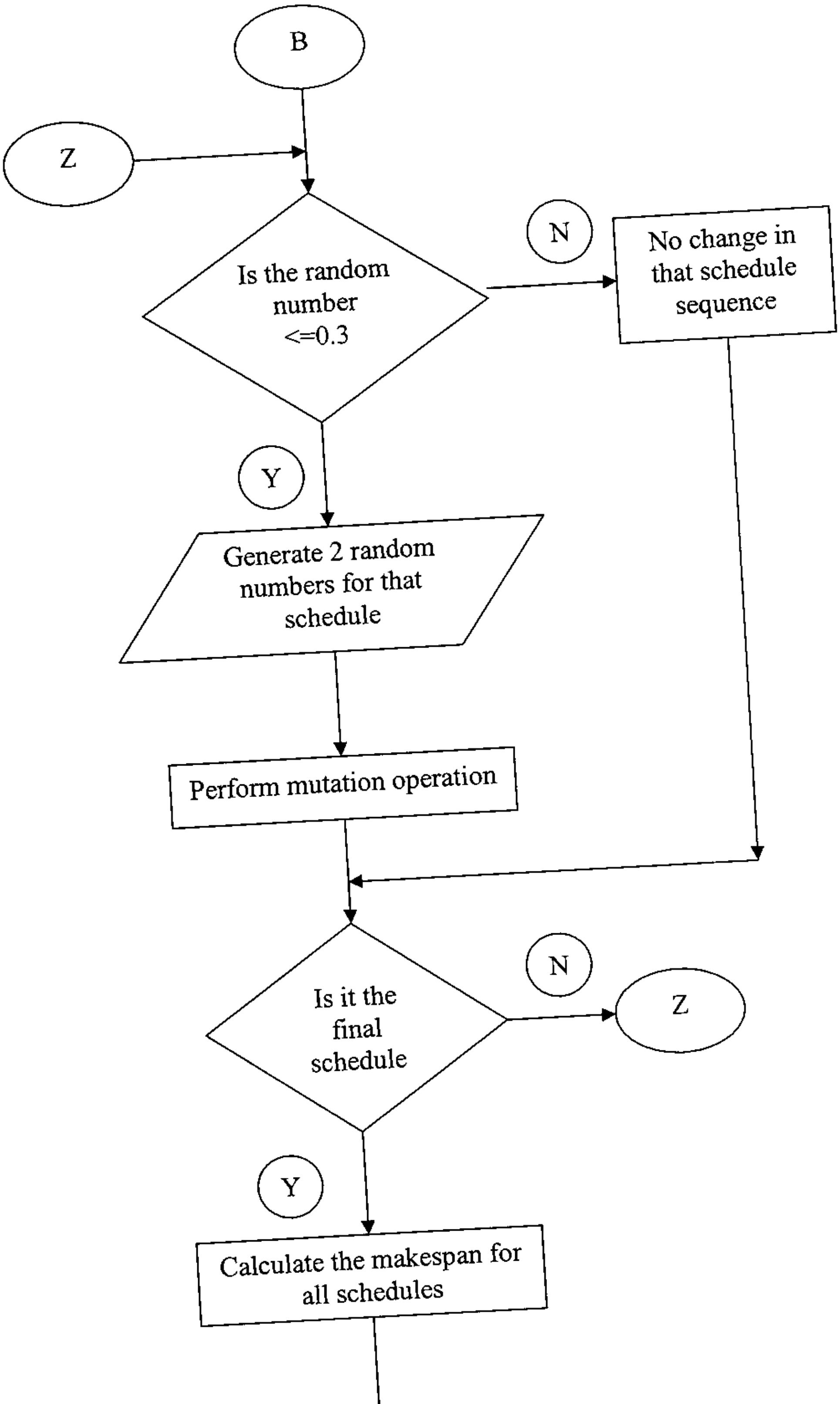
Generate initial population
Random population evaluation
Reproduction
Crossover
Mutation
New population
N
Is the criterion satisfied?
Y
Print the optimum solution

5.2 PROGRAM FLOW CHART



Generate random numbers between 0 to 9 using rand() function for 30 schedules
Is same number repeating
Y
Drop that number and take the next number
N
Calculate the makespan for each schedule
T
Sort the makespan in descending order
Rank the sorted makespan from 1 to 30
Generate 15 random numbers between 0 to 9 for each pair of schedule with 0.1 unit of accuracy
A


A
Is the random number <=0.6
X
N
No change in the pair
Y
Select the pair for crossover
Perform crossover operation
Is it the last pair of schedule
N
X
Y
Generate random numbers between 0 to 9 with 0.01 unit accuracy for all schedules
B


B
Z
Is the random number <=0.3
N
No change in that schedule sequence
Y
Generate 2 random numbers for that schedule
Perform mutation operation
Is it the final schedule
N
Z
Y
Calculate the makespan for all schedules

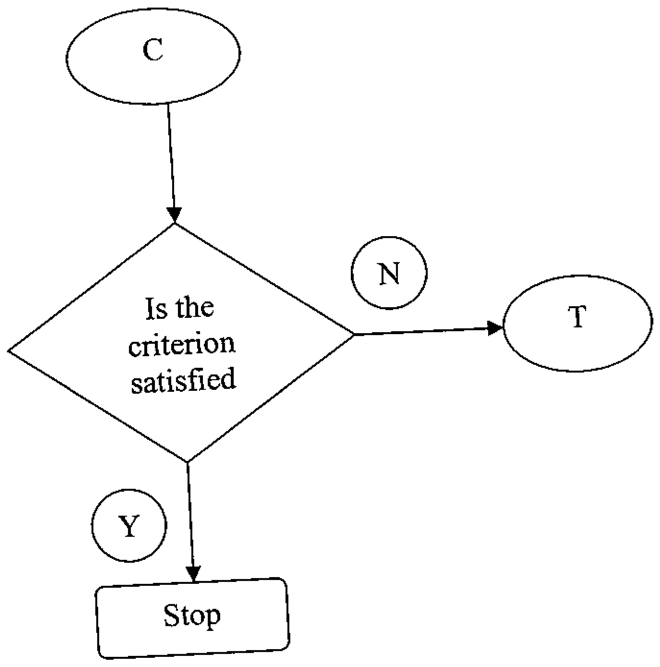


FIGURE 5.2 PROGRAM FLOW CHART

In Figure 5.2, the function rand () returns a positive integer, which is used to generate random number schedules. Makespan is calculated for each schedule, generated using the rand () function. Rank Selection (sorting) is done to eliminate the weak schedules i.e. the schedules with maximum makespan and to produce strong schedules i.e. the schedules with minimum makespan. The random numbers 0.3 and 0.6 are the probability of mutation (P_m) and probability of crossover (P_c) respectively. In the process of crossover, a pair of schedule is considered to perform crossover operation. If the random number is less than 0.6 crossover operation is performed for that pair of schedule (refer section 4.2.2), else next pair is taken. In the process of mutation, single schedule is taken to perform mutation operation. If the random number is less than 0.3 mutation operation is performed for each schedule (refer section 4.2.3), else next schedule is taken to perform mutation operation. In this way, more number of iterations are performed until a minimum makespan is obtained. The makespan which occurs

CHAPTER 6

RESULTS AND DISCUSSION

For the flow shop scheduling problem defined in chapter 3, a C program was developed according to the flow chart described in figure 5.2. The following are the assumptions made to solve this flow shop scheduling problem,

i. A set of 10 jobs is available at time zero.

ii. The two machines are continuously available.

iii. Idle time is not taken into account.

iv. A job entering machine 1 leaves machine 2 after completing all the operations on two machines.

TABLE 6.1 MAKESPAN CALCULATION

The makespan for the schedule 8-0-1-2-6-4-5-9-7-3 is shown below in the following table.

JOBS	MACHINE 1 (TIME IN)	MACHINE 1 (TIME OUT)	MACHINE 2 (TIME IN)	MACHINE 2 (TIME OUT)
8	0	3000	3000	6000
0	3000	5000	6000	8000
1	5000	8000	8000	10000
2	8000	9000	10000	11000
6	9000	10500	11000	12500
4	10500	12000	12500	15500
5	12000	14000	15500	17500
9	14000	16000	17500	18500
7	16000	17250	18500	21000

The job 8 enters machine 1 at time 0 since it is the first job of the schedule. It's processing time t_{11} is 3000 minutes i.e. Processing time (30) * Batch Quantity (100) = 3000 (refer Table 3.1). t_{11} is the processing time of first job (job 8) on machine 1. Job 0 enters machine 1 at 3000 minutes after job 8 has completed its operation on machine 1. The second job (job 0) enters machine at 6000 minutes instead of 5000 minutes (t_{21}) since the first job completes its operation on machine at 6000 minutes. In this way, the makespan is calculated for the whole sequence. The makespan for this schedule is 22000. Table 6.2 displays the makespan for the schedules generated using random number generation. Table 6.3 shows the result of first iteration.

TABLE 6.2 RANDOM POPULATION EVALUATION

S.NO.	SCHEDULE SEQUENCE	MAKESPAN
1	4-1-0-2-6-5-3-9-7-8	22250
2	8-0-9-4-6-1-5-3-7-2	22000
3	0-9-8-1-5-4-6-2-3-7	23000
4	9-0-5-1-6-3-4-2-8-7	23500
5	8-9-2-0-6-5-1-3-5-7	23250
6	0-8-2-9-6-4-1-3-5-7	22500
7	0-7-3-9-6-5-4-2-8-1	21750
8	6-0-1-4-3-8-2-7-5-9	22500
9	8-5-2-3-4-1-0-6-7-9	22000
10	6-9-5-8-2-4-3-1-7-0	23000
11	8-2-0-1-4-3-9-7-6-5	22000
12	1-0-5-4-2-6-3-7-9-8	22250
13	0-6-5-1-7-2-4-3-9-8	22250
14	0-6-1-3-4-5-8-7-9-2	22500
15	6-3-0-1-7-4-8-2-5-9	23000
16	0-8-3-2-1-4-6-7-9-5	23000

18	1-4-5-6-8-0-2-9-3-7	22000
19	9-0-4-2-7-8-6-1-3-5	22000
20	9-2-7-8-4-5-3-6-0-1	21750
21	6-8-0-1-2-4-5-3-9-7	22000
22	6-0-1-8-4-5-2-9-3-7	23000
23	0-5-1-2-3-4-7-6-9-8	22500
24	9-6-1-0-7-8-3-2-5-4	23000
25	0-5-4-2-9-8-6-3-1-7	22500
26	0-2-9-5-4-3-1-8-7-6	23500
27	0-5-4-8-6-3-7-2-1-9	**21000**
28	6-2-5-0-8-4-1-7-3-9	22000
29	8-4-6-0-2-1-3-9-5-7	22500
30	8-0-6-2-5-1-3-4-9-7	22500

TABLE 6.3 RESULT OF FIRST ITERATION

JOBS	SCHEDULE SEQUENCE	MAKESPAN
1	0-1-4-2-6-5-3-9-7-8	22250
2	8-0-9-4-6-1-5-7-3-2	22000
3	0-5-1-9-4-3-6-2-8-7	23500
4	9-8-0-1-5-2-6-4-3-7	23000
5	8-9-2-4-0-1-3-5-6-7	22500
6	0-8-6-5-9-1-3-2-7-4	23250
7	0-5-3-9-6-7-4-2-8-1	22500
8	6-0-1-4-3-8-2-7-5-9	22500
9	8-3-2-5-4-1-0-6-7-9	22000
10	9-6-5-8-2-4-3-1-70	23000
11	0-2-8-1-4-3-9-7-6-5	22000
12	10-5-4-2-6-3-7-9-8	22250
13	0-6-1-3-4-5-8-7-9-2	22500

16	0-8-3-2-1-4-5-3-7-9	23000
17	4-5-8-3-0-1-9-7-6-2	21750
18	1-4-5-0-8-6-2-9-3-7	22000
19	0-9-4-5-2-3-6-8-1-7	22500
20	9-2-7-8-0-6-1-4-3-5	21750
21	0-8-6-4-1-5-2-9-3-7	22000
22	6-0-1-8-2-4-5-3-9-7	23000
23	0-5-1-2-3-4-7-6-9-8	22500
24	3-6-1-0-7-8-9-2-5-4	23000
25	0-5-4-3-1-8-7-9-2-6	22500
26	0-2-9-8-6-3-1-7-4-5	24000
27	0-5-4-8-6-1-2-7-3-9	**21000**
28	6-2-5-4-8-7-3-0-1-9	**21000**
29	8-4-6-0-2-1-3-9-5-7	22500
30	8-0-6-2-5-1-3-4-9-7	22500

Before using GA the minimum makespan is 21000 minutes. After using GA i.e. after the operations such as rank selection, crossover and mutation the makspan is 21000 minutes, but it has been obtained for two schedules 0-5-4-8-6-1-2-7-3-9 and 6-2-5-4-8-7-3-0-1-9. From this, it is to be understood that a weaker schedule was eliminated and a strong schedule has been produced more than once.

Since random number generation is involved in this project work, the program is written in such a way that the result of first iteration is taken into account for second iteration, for second iteration third iteration is taken into account. In this way 20 iterations are performed, finally minimum makespan is obtained at 20th iteration. Table 6.4 shows the results of 20 iterations obtained using C program.

TABLE 6.4 RESULTS OF 20 ITERATIONS

ITERATION NO.	SCHEDULE SEQUENCE	MINIMUM MAKESPAN
1	0-5-4-8-6-1-2-7-3-9	21000
2	0-4-8-1-5-9-6-2-3-7	21750
3	7-8-0-4-9-2-6-1-5-3	20750
4	7-8-0-4-6-9-5-3-2-1	21250
5	7-5-0-4-6-1-9-2-8-3	21250
6	4-0-7-8-6-5-1-9-3-2	20500
7	4-0-7-8-6-5-1-3-9-2	20500
8	4-0-1-7-8-6-5-3-9-2	20500
9	7-5-6-9-4-8-1-0-2-3	20250
10	4-0-7-3-8-6-5-1-9-2	20500
11	6-4-7-5-8-9-1-0-2-3	20500
12	2-4-7-5-8-9-1-0-6-3	20500
13	2-4-7-5-8-9-1-0-6-3	20500
14	3-7-0-2-4-8-1-5-6-9	21250
15	6-2-3-4-0-8-7-5-1-9	`21500
16	2-4-1-5-7-6-8-9-0-3	20500
17	7-6-5-4-1-8-2-0-3-9	20250
18	7-8-5-0-6-2-4-3-1-9	20750
19	2-7-4-1-8-5-6-0-9-3	20250
20	**2-7-4-1-8-5-6-0-9-3**	**20250**

From the Table 6.4, 20 iterations have been performed, makespan 20500 and 20250 occurs repeatedly. Even though 20500 occurs more than 20250, the minimum makespan is taken as the optimum result. But four schedule sequences have the same makespan as 20250. i.e. the schedules at iterations at 9, 10, 19 and **2-7-4-1-8-5-6-0-9-3** is selected as the optimum result, since it

CHAPTER 7

CONCLUSION

A Flow shop scheduling problem of 2 (m) machines and 10 (n) jobs has been solved using Genetic Algorithm with the objective of obtaining minimum makespan. Random number schedules are generated and an optimum result of minimum makespan is found after undergoing the operations such as reproduction, crossover and mutation. The makespan is 20250 minutes for the case study problem and the schedule is 2-7-4-1-8-5-6-0-9-3. The optimization is reached at the 20th iteration as 19th and 20th iteration yield same makespan time. The same program could be used to find the makespan for more than two machines and 10 jobs problems also by modifying a simple logic in makespan calculation.

Further Work:

In addition to makespan minimization, the penalty cost could also be considered for optimizing the scheduling problem.

APPENDIX

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void makespan(int apan[][10],long panmin[]);
void genran(int arr[][10]);
long  sort(long pans[]);
void replace(int arr[][10],int rowj,int ran);
void interchange(int arr[][10],int rw,int ran1,int ran2);
void main()
{
int t,rg,aa,a[50][10],j,i,ranval,row,ranval1;
long panswer[30],makespans;
float ra[30],r[30],r1,n1=0.01;
int fst[9],sec[9],x,y,ss,ss1,n=5;
genran(a); // To Generate Random Numbers
makespan(a,panswer);//To Find Makespan
/*for(j=0;j<=29;j++)
        {       for(i=0;i<=9;i++)

                { printf("%d ",a[j][i]);}
        printf(" %ld", panswer[j]);
        printf("\t ");
        }*/
makespans=sort(panswer);//Rank Selection
//First
//printf("The makespan of first iteration is %ld\n",makespans);
for(i=0;i<15;i++)
        {r[i]=((rand()%10)*0.1);}//Random Number Generation for Cross over
for(i=0;i<30;i++)
        {ra[i]=((rand()%10)*.02)+.2; }//Random Number Generation for Mutation
for(i=0;i<30;i++)
{
//aa=((rand()%10)*.02)+0.2;
//printf("%.2f ",ra[i]);
}
row=0;

for(rg=0;rg<15;rg++)
{

                if(r[rg]<=0.6)
                {
                        ranval=((rand()%10)*rg);
                        if(ranval>=100) {ranval=ranval%100;}
        //              printf("ran =%d ",ranval);
                        replace(a,row,ranval);//Cross Over
                  }
row=row+2;
}
printf("\n");
/*for(j=0;j<=29;j++)
        {       for(i=0;i<=9;i++)
                 {printf("%d ",a[j][i]);}
        printf("\t%ld\n ", panswer[j]);
                 }
  */
 row=0;
 for(rg=0;rg<30;rg++)
 {
                  if(ra[rg]<=0.3)
                  {
                        ranval=((rand()%10)*rg) ;
                        ranval1=((rand()%10)*rg);
                        if(ranval>=100) {ranval=ranval%100;}
                        if(ranval>=10) {ranval=ranval%10;}
                        if(ranval1>=100) {ranval1=ranval1%100;}
                        if(ranval1>=10) {ranval1=ranval1%10;}
                        if(ranval==0) {ranval=ranval+4;}
                        if(ranval1==0){ranval1=ranval1+2;}
                      // if(ranval==9) {ranval=ranval-1;}
                      // if(ranval1==9) { ranval1=ranval1-1;}
    //                  printf("%d\t",ranval);
      //                printf("%d\t",ranval1);
  //                      { ranval=ranval+1;ranval1=ranval1+1; }

                        interchange(a,row,ranval,ranval1);//Mutation
          //            printf("%d\t",ranval);
            //          printf("%d\t",ranval1);


                  }
   row=row+1;
  }
  //printf("\n");
  makespan(a,panswer);
  /*for(j=0;j<=29;j++)
          {       for(i=0;i<=9;i++)
                   {printf("%d ",a[j][i]);}
          printf("\t%ld\n ", panswer[j]);
                   }*/
   makespans=sort(panswer);
   //printf("The makespan of next iteration is %ld",makespans);
```

```
for(i=0;i<15;i++)
        {r[i]=((rand()%10)*0.09);}
//      printf("%.2f ",r[i]);printf("\n"); }//Random Number Generation for Cross
over
for(i=0;i<30;i++)
        {ra[i]=((rand()%10)*.03)+.1;}
 //     printf("%.2f ",ra[i]); }//Random Number Generation for Mutation
row=0;

 for(rg=0;rg<15;rg++)
 {

                if(r[rg]<=0.6)
                {       n=n+2;
                        ranval=((rand()%10)*n);
                        if(ranval>=100) {ranval=ranval%100;}
                        if(ranval>=10) {ranval=ranval%10;}
                        if(ranval==0) {ranval=ranval+2;}
                        if(ranval==9){ranval=ranval-3;}
            //          printf("ran =%d ",ranval);
                        replace(a,row,ranval);//Cross Over
                }
  row=row+2;
  }
  printf("\n");
  n=3;
  row=0;
  for(rg=0;rg<30;rg++)
  {
                if(ra[rg]<=0.3)
                {
                n=n+5;
                        ranval=((rand()%10)) ;
                        ranval1=((rand()%10));
                        if(ranval>=100) {ranval=ranval%100;}
                        if(ranval>=10) {ranval=ranval%10;}
                        if(ranval1>=100) {ranval1=ranval1%100;}
                        if(ranval1>=10) {ranval1=ranval1%10;}
                        if(ranval==0) {ranval=ranval+2;}
                        if(ranval1==0){ranval1=ranval1+4;}
                        interchange(a,row,ranval,ranval1);//Mutation
                 }
   row=row+1;
  }
  makespan(a,panswer);
  for(j=0;j<=29;j++)
        {       for(i=0;i<=9;i++)
```

```
makespans=sort(panswer);
printf("The makespan of next iteration is %ld",makespans);

 getch();
 clrscr();
 }

 void interchange(int arr[][10],int rw,int ran1,int ran2)
 {
 int t;
 t=arr[rw][ran1-1];
 arr[rw][ran1-1]=arr[rw][ran2-1];
 arr[rw][ran2-1]=t;
 }

  void replace(int arr[][10],int rowj,int ran)
  {
  int m,i,n,x=0,y=0,acr,acr1,first[15],second[15];
            for(m=0;m<=9;m++)
            {
            for(n=ran;n<=9;n++)
                      {
                      if(arr[rowj+1][m]==arr[rowj][n])
                                { first[x]=arr[rowj][n];x++; }
                      if(arr[rowj][m]==arr[rowj+1][n])
                                { second[y]=arr[rowj+1][n];y++; }

                       }
              }
   //         first[x]=0;
    //        second[x]=0;
   //x=0;
   acr=ran;
              for(i=0;i<x;i++)
              {
               arr[rowj][acr]=first[i];
               arr[rowj+1][acr]=second[i];
               acr++;
               }
    /*        for(i=ran;i<=10;i++)
               {
               arr[rowj][i]=first[x];
               arr[rowj+1][i]=second[x];
     //        arr[rowj+1][10]=0;
     x++;
     } */
     }
```

```
long pan[9],x,i,j,fsum=0;
int f1[]={2000,3000,1000,2000,1500,2000,1500,1250,3000,2000};
int s1[]={2000,2000,1000,1000,3000,2000,1500,2500,3000,1000};
        for(j=0;j<=29;j++)
          {
                fsum=0;
        for(i=0;i<=9;i++)
                {
                x=apan[j][i];
                fsum=fsum+f1[x];
                if(i==0)
                   {pan[i]=fsum+s1[x];}
                else if(fsum<pan[i-1])
                   {pan[i]=pan[i-1]+s1[x];}
                else
                   {pan[i]=fsum+s1[x];}
                 }
              //  apan[j][10]=0;
  panmin[j]=pan[9];
  //pan[10]=1;
          }
  }

  long sort(long pans[])
  {
   int i,j,t;
          for(i=0;i<=28;i++)
          {
           for(j=i+1;j<=29;j++)
            {
                if(pans[i]<pans[j]) //sorting in Descending order
                { t=pans[i];
                  pans[i]=pans[j];
                  pans[j]=t;
                }
            }
          }
   printf("\n");
   return(pans[29]);
   }

   void genran(int arr[][10])
   {
   int a,count,i,j,r,s;
   int x,m;
           for(j=0;j<=29;j++)
           {
            {       if(j>=i)
                     {a=(rand()%10)* (j-i);}
                     else
                     {a=(rand()%10)*(i-j);}
            }
            else
            {a=(rand()%10)* (j-i);}
            if (a>99)
                     a=a%100;
            if(a>9 && a<=99)
                     a=a%10;
            arr[j][i]=a;
        }
}

         for(j=0;j<=29;j++)
                  {

                   x=0; count=0;
                  while(x<=9)
                  {
                   for(i=0;i<=9;i++)
                            {
                            if(arr[j][i]==x)
                                      {count++;goto nn;}
                            }
                   if(count==0)
                   {
                            for(r=0;r<=8;r++)
                             {
                             for(s=r+1;s<=9;s++)
                               {
                               if(arr[j][r]==arr[j][s])
                                       { arr[j][s]=x;goto nn;}
                                }
                              }
                    }

        nn:  count=0;x++;
              }
       }
    }
```

REFERENCES

1. Aggoune, R., (2004), Minimizing the Makespan for the Flow Shop Scheduling Problem with the available Constraints, *European Journal of Operational Research,* http//: www.scirus.com.
2. Allahverdi, A., (2003), The Two- and M-machine Flowshop Scheduling Problems with Bicriteria of Makespan and Mean Flowtime, *European Journal of Operational Research,* http//:www.scirus.com.
3. David, E., Goldeberg, (2000), *Genetic Algorithm in Search, Optimization and Machine Learning,* pp. 1-18, 60-70.
4. Deitel & Deitel, (2001), *C How to Program,* pp. 100-128, 144-155.
5. Montanna, D., Brinn, M., Moore, S., Bidwell, G., Genetic Algorithms for Complex, Real Time Scheduling, *BBN Technologies/GTF Networking, Cambridge.*
6. Gopala Krishnan, A., Dr.K.Mallikarjuna Rao,(2003), Optimization of Job-Shop Scheduling Using Non-Traditional Optimization Techniques, *Industrial Engineering Journal,* pp. 25-28.
7. Iyer, S.K., Saxena, B., (2004), Computers and Operations Research, Improved Genetic Algorithm for the Permutation Flowshop Scheduling Problem, http//:www.sciencedirect.com.
8. Baker, K.R., (1974), Duke University, *Introduction to Sequencing and Scheduling,* pp. 136-173, 178-205.
9. Deb, K., (2002), *Optimization for Engineering Design, Algorithms*
10. Neppalli, V.R., Chuen-Lung, C., and Gupta, J.N.D., (2006),Genetic Algorithms for the Two-Stage Bicriteria Flowshop Problem, *European Journal of Operational Research*, http//:www.sciencedirect.com
11. Portmann, M.C., Vignier, A., Dardilhac, D., Dealer, D., (1998), Branch and Bound Crossed with GA to Solve Hybrid Flowshops, *European Journal of Operational Research*, http//:www.sciencedirect.com
12. Raja Kumar, S., Arunachalam, V.P., Selladurai, V., (2004), Workflow Balancing Strategies in Parallel Machine Scheduling, *International Journal Of Manufacturing Technology*

13. Ravichandran, K.S., Chandra Sekhara Rao, K., Saravanan, R., (2002), The Role of Fuzzy and Genetic Algorithms in Part Family Formation and Sequence Optimization for Flexible Manufacturing Systems, *International Journal On Manufacturing Technology.*

14. Rajasekaran S., Vijayalakshmi Pai G.A., (2005), *Neural Networks, Fuzzy Logic and Genetic Algorithm,* pp. 225-252, 253-293.

15. Ruiz, R., Maroto, C., Alcaraz, J., (October 2006), Two New Robust Genetic Algorithms for Flow Shop Scheduling Problem, *European Journal of Operations Research,* http//: www.scirus.com.

16. Yashavant Kanetkar, (2000), *Let Us C,* pp. 1-123, 157-200, 275-310.