

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

**A GENETIC ALGORITHM FOR A FLEXIBLE
JOB SHOP SCHEDULING PROBLEM WITH
SEQUENCE DEPENDENT SETUP TIMES**

by
Ezgi ÖZDÖL

October, 2011
İZMİR

**A GENETIC ALGORITHM FOR A FLEXIBLE
JOB SHOP SCHEDULING PROBLEM WITH
SEQUENCE DEPENDENT SET UP TIMES**

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
In Industrial Engineering Program**

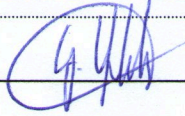
**by
Ezgi ÖZDÖL**

**October, 2011
İZMİR**


M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “A GENETIC ALGORITHM FOR A FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SET UP TIMES” completed by EZGİ ÖZDÖL under supervision of ASST. PROF. DR. GÖKALP YILDIZ and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Gökalp YILDIZ



Supervisor



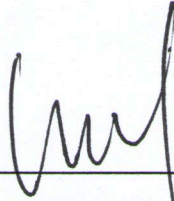
Assoc. Prof. Dr. Seydi TOPALOĞLU

(Jury Member)



Asst. Prof. Dr. Güvener K. DEMİR

(Jury Member)



Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor Asst. Prof. Dr. Gökalp YILDIZ for his guidance and patience to complete my thesis.

I thank my manager and colleagues who support me in both academic and general aspects.

Finally, I would like to give the deepest thanks to my family for their love and endless supports. It would have been impossible for me to complete this research without their love and encouragement.

Ezgi ÖZDÖL

A GENETIC ALGORITHM FOR A FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SET UP TIMES

ABSTRACT

This thesis addresses the problem of scheduling flexible job shops with sequence dependent setup times, and aims at minimizing makespan as an objective by using genetic algorithms. The flexible job shop is characterized by job flow through a number of work centers that contain identical or nonidentical parallel machines. Flexible Job Shop Problems divided into two subproblems. In the first subproblem, assignments of operations to machines are made. In the second subproblem, sequencing problem is solved. For the first subproblem, three assignment methods were used in this thesis. The first method is “Approach by Localization” which assigns each operation to the suitable machine by taking into accounts the processing times and workloads of machines. This assignment procedure assumes that there are no setup times. The second method is modified version of the first method. In this method, we modified the assignment procedure by using sequence dependent setup times in addition to processing times and workloads of machines. In the third method, assignments were made randomly just for the comparison purposes. For the second subproblem, dispatching rules such as shortest processing time (SPT), most work remaining (MWR), longest processing time (LPT), shortest setup time first rule (SSTFR), random rule and neighborhood search were applied to sequence the machines.

The main feature of this thesis is to combine the sequence dependent setup times with the processing times and the workloads on both assignment and sequencing procedures. The performance of genetic algorithm was also analyzed by taking into account the issues such as, initial population generation, sequencing, selection and mutation methods.

Keywords: Flexible Job Shop Scheduling, Sequence Dependent Set-up Time, Genetic Algorithms

SIRA BAĞIMLI HAZIRLIK ZAMANLI ESNEK ATÖLYE TİPİ ÇİZELGELEME PROBLEMİ İÇİN BİR GENETİK ALGORİTMA

ÖZ

Bu tezde, sıra bağımlı hazırlık zamanlı esnek atölye çizelgeleme problemlerinden, tamamlanma zamanı minimizasyonu Genetik Algoritma ile çözülmüştür. Esnek atölye çizelgeleme problemlerinde iş akışı paralel ve paralel olmayan iş istasyonları arasında olmaktadır. Esnek atölye çizelgeleme problemi iki alt problemden oluşmaktadır. İlk problem; her bir operasyonun makinalara atamaların yapılması, ikinci problem ise ataması yapılan makina önündeki iş sıralamalarının yapılmasıdır. Bu tezde, ilk problemde, üç makine atama yöntemi kullanılmıştır. İlk yöntem her bir operasyonu makinalara atarken makina yüklerini ve iş sürelerini göz önünde bulundurarak atama yapan “Approach by Localization” yöntemidir. Bu yöntem makine hazırlık zamanlarını göz önünde bulundurmamaktadır. İkinci yöntem “Approach by Localization” yönteminin sıra bağımlı hazırlık zamanlarını da makina yükleri ve iş süreleri ile birlikte göz önünde bulundurularak modife edildiği yöntemdir. Üçüncü yöntem ise karşılaştırma amaçlı kullanılan, makine atamalarının rastgele yapıldığı yöntemdir. İkinci problemde operasyon ataması yapılmış her bir makina önündeki sıralamalar, En Kısa Hazırlık Zamanı, En Kısa İşlem Zamanı, En Uzun İşlem Zamanı, En Uzun Kalan Süre ve Rasgele, öncelik dağıtım kuralları ile oluşturulmaktadır.

Bu tezde, makine atamaları ve makine önündeki sıralamalar yapılırken sıra bağımlı hazırlık zamanları iş süreleri ve makine yükleri ile birlikte ele alınmıştır. Farklı problem büyüklüklerinde başlangıç popülasyonu yaratma, sistemde kalma süresi hesaplama, seçme ve mutasyon yöntemlerinin genetik algoritma performansı üstünde etkileri araştırılmıştır.

Anahtar Sözcükler: Esnek Atölye Çizelgeleme, Sıra Bağımlı Hazırlık Zamanı, Genetik Algoritmalar

CONTENTS

	Page
THESIS EXAMINATION RESULT FORM	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZ	v
CHAPTER ONE-INTRODUCTION	1
1.1 Motivation of the Research	1
1.2 Thesis Outline	3
CHAPTER TWO-SCHEDULING PROBLEMS.....	4
2.1 Introduction	4
2.2 Notation of Scheduling Problems	5
2.2.1 Machine Environment (α).....	6
2.2.1.1 Single Machine.....	7
2.2.1.2 Parallel Machine.....	7
2.2.1.3 Flow Shop	7
2.2.1.4 Job Shop	8
2.2.1.5 Flexible Flow Shop	8
2.2.1.6 Flexible Job Shop	9
2.2.2 Side Constraints (β)	9
2.2.3 Objective (γ).....	11
2.2.3.1 Performance Measures Based on Completion Time	11
2.2.3.2 Performance Measures Based on Flow Time.....	12
2.2.3.3 Performance Measures Based on Lateness	13
2.2.3.4 Performance Measures Based on Number of Late/tardy Jobs	15
2.2.3.5 Performance Measures Based on Tardiness.....	15

2.2.3.6 Performance Measures Based on Earliness.....	16
2.3 General Assumption of Scheduling Problems	18
2.4 Summary	19
CHAPTER THREE-FLEXIBLE JOB SHOP SCHEDULING.....	20
3.1 Introduction.....	20
3.2 Solution Approaches for Scheduling Problems	21
3.2.1 Exact Optimisation Methods.....	22
3.2.1.1 Dynamic Programming	22
3.2.1.2 Integer Programming	22
3.2.2 Heuristic Approaches.....	23
3.2.2.1 Dispatching Rules	23
3.2.2.2 Local Search.....	23
3.3 Literature Review.....	26
3.4 Summary	36
CHAPTER FOUR-GENETIC ALGORITHMS.....	37
4.1 Introduction.....	37
4.2 Solving Scheduling Problems with Genetic Algorithms	37
4.3 Genetic Algorithm.....	38
4.3.1 Chromosome Representation	39
4.3.2 Genetic Operators	42
4.3.2.1 Selection Operator.....	43
4.3.2.1.1 Roulette-Wheel Selection	44
4.3.2.1.2 Rank Based Selection.....	44
4.3.2.1.3 Tournament Selection	44
4.3.2.2 Crossover Operators.....	45
4.3.2.2.1 Partial-Mapped crossover (PMX)	45
4.3.2.2.2 Order Crossover (OX).....	45
4.3.2.2.3 Position-Based Crossover	46
4.3.2.2.4 Order-Based Crossover	46
4.3.2.2.5 Cycle Crossover (CX).....	46
4.3.2.2.6 Linear Order Crossover (LOX).....	47

4.3.2.2.7 Subsequence Exchange Crossover	47
4.3.2.2.8 Job-Based Order Crossover	48
4.3.2.2.9 Partial Schedule Exchange Crossover.....	48
4.3.2.2.10 Substring Exchange Crossover	49
4.3.2.3 Mutation Operator	49
4.3.3 Genetic Parameters	50
4.4 Summary	51
CHAPTER FIVE-SOLVING FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SET UP TIME BY USING GENETIC ALGORITHM	52
5.1 Introduction.....	52
5.2 Description of the Problem	53
5.3 Performance Measures	55
5.4 Proposed Genetic Algorithm.....	55
5.5 Initial Population Generation	56
5.5.1 Fitness Evaluation	64
5.5.1.1 Dispatching Rules	64
5.5.1.1.1 Most Work Remaining Rule (MWR).....	65
5.5.1.1.2 Shortest Processing Time Rule (SPT).....	65
5.5.1.1.3 Longest Processing Time Rule (LPT)	65
5.5.1.1.4 Random Rule.....	65
5.5.1.1.5 Shortest Setup Time First Rule (SSTFR).....	65
5.5.1.2 Neighborhood Search.....	66
5.5.2 Selection.....	66
5.5.2.1 Selection of Reproduction.....	66
5.5.2.2 Selection of Replacement.....	67
5.5.3 Crossover	67
5.5.4 Mutation.....	69
5.6 Parameters	72
5.7 Computational Results	74

CHAPTER SIX-CONCLUSION	79
6.1 Conclusion	79
6.2 Future Research.....	80
REFERENCES.....	81
APPENDICES	85

CHAPTER ONE

INTRODUCTION

1.1 Motivation of the Research

Many modern manufacturing environments are flexible job shops. Advanced machines allow different jobs to be processed on the same machine by changing machine tools. Setup activities of a new job then depend on the previous job on the machine. Some complicated products require multiple visits to the same processing station and all jobs are not ready to start their processing operation at the same time.

This thesis is related to reentrant flexible job shop with sequence dependent setup time. A reentrant flexible job shop refers to the machine environment in which each job should be processed on each one of a series of work centers has its own predetermined route to follow, and may visit a machine more than once.

Scheduling reentrant flexible job shops with the objective of minimizing makespan is one of the most complex scheduling problems and has been proved to be NP-hard. There are several researchers who developed methods to solve this problem. Processing restrictions, such as sequence-dependent setup times and recirculation have received more attention from many researchers ((Kacem, Hammadi and Borne, 2003); (Chen and Wu, 2007); (Pezzella, Morganti and Ciaschetti, 2008); (Gao, Sun and Gen, 2008); (Gen, Gao and Lin, 2009); (Fattahi and Fallahi, 2010)).

In order to specify assumptions for this thesis, we briefly describe the characteristics of the problem in the following list.

1. Each job consists of one fixed sequence of operations.
2. Each machine can process at most one job at a time, and each job can only be processed on one machine at a time.

3. There are no interruptions and cancellations between jobs.
4. All machines are available continuously.
5. There are identical or nonidentical parallel machines
6. All jobs can be started at $t = 0$.
7. All machines are available at $t = 0$.
8. There are sequence dependent setup time between operations.
9. Setup for a job cannot begin until the job is available to the current work center and the desired machine in the work center is idle.
10. All data including processing times and setup times are deterministic.
11. There are no precedence constraints among operations of different jobs.
12. Neither release times nor due dates are specified.
13. There are two cases, the first is that all machine can process all the jobs (total flexibility), and the other is that all machines can not process all the jobs (partial flexibility).

The significance of this thesis can be summarized as follows:

1. This thesis takes into consideration flexible job shop problems including identical or nonidentical parallel machines and sequence-dependent setup times. These characteristics usually occur in real world production environments.
2. Genetic Algorithm is used for solving the flexible job shop scheduling problem with the sequence dependent setup time.
3. Test problems of different sizes have been solved and regression analysis has been used for showing the effects of makespan calculation, initial population generation, selection and mutation methods on the genetic algorithm performance.

1.2 Thesis Outline

In Chapter Two, the classification of problem is given. In Chapter Three, the literature review of flexible job shop scheduling problems is given. In Chapter Four, basic information about genetic algorithm is given, In Chapter Five, the design and the implementation issues of genetic algorithm for flexible job shop scheduling problems with sequence dependent setup time are discussed. In Chapter Six, conclusions and the future research is given.

CHAPTER TWO

SCHEDULING PROBLEMS

2.1 Introduction

Scheduling is a decision making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods, and its goal is to optimize one or more objectives (Pinedo, 2002).

Baker (1992) categorizes the major scheduling models by specifying the resource configuration and the nature of tasks. The resources and tasks in an organization can take many different forms. The resources may be machines and the tasks may be operations in a production process.

Pinedo (2002) describes the scheduling problems according to four types of information:

1. The jobs and operations to be processed.
2. The number and types of machines that the shop comprises.
3. Disciplines that restrict the manner in which assignments can be made.

4. The criteria by which a schedule will be evaluated.

The nature of job arrivals provides the distinction between static and dynamic problems. If a set of jobs available for scheduling does not change over time, the system is called as static, in contrast to cases in which new jobs appear over time, where the system is called as dynamic. Traditionally, static models have proven to be more tractable than dynamic models, and have been studied more extensively (Baker, 1992).

Two kinds of feasibility constraints are commonly found in scheduling problems. First, there are limits on the capacity of machines, and the second, there are technological restrictions on the order in which some jobs can be performed

Scheduling functions rely on mathematical techniques and heuristic methods to allocate limited resources to the activities that must be done. This allocation of resources must be done in such a way that the company optimizes its objectives and achieves its goals. Objectives may have many different forms, such as minimizing the time needed for completing all activities, minimizing the number of activities that are completed after the committed due dates and so on (Pinedo, 2002).

2.2 Notation of Scheduling Problems

Scheduling problem descriptions include number of abbreviations and symbols that represent characteristics and functions or variables. Graham (1979) classifies machine scheduling problems by a standard three-field notation. This notation is represented by $\alpha | \beta | \gamma$ which is generic notation of scheduling problems. First field “ α ” is used to describe the machine environment. This field allows for the identification of single machine, various types of parallel machines, flow shop, job shop, flexible job shop, and flexible flow shop problems. The second field “ β ” is used for describing the task and resource characteristics. This field includes parameter settings for characterizing the possibility for release dates, precedence constraints, sequence dependent setup times, preemptions, storage / waiting constraints, machine eligibility and recirculation. The third field “ γ ” is used for denoting the performance measures. This field includes completion time, flow time, tardiness and earliness, etc.

α : Machine Configuration

(1) Single-Machine

(P_m) Identical Parallel-Machine

(R_m) Unrelated Parallel Machine

- (F_m) Flow Shop
- (J_m) Job Shop
- (FF_c) Flexible Flow Shop
- (FJ_c) Flexible Job Shop

β : Constraints

- (r_j) Release dates
- (prec) Precedence constraints
- ($s_{j,k}$) Sequence dependent setup times
- (prmp) Preemptions (resume or repeat)
- (block) Storage / waiting constraints
- (M_j) Machine eligibility
- (circ) Recirculation

γ : Objectives and Performance Measures

1. Completion time
2. Flow time
3. Tardiness
4. Earliness
5. Lateness
6. Throughput
7. Late/tardy jobs

2.2.1 Machine Environment (α)

The possible machine environments are single machine, parallel machine, flow shop and job shop models.

2.2.1.1 Single Machine

In this case, there is only one machine ($\alpha = 1$). The basic single machine model is fundamental in the study of sequencing (Pinedo, 2002). The simplest sequencing problem is the one in which there is a single resource or machine. Single machine models are also important in decomposition methods, when scheduling problems in more complicated machine environments are divided into a number of smaller single machine scheduling problems.

2.2.1.2 Parallel Machine

Parallel machine model is a generalization of the single machine model. Many production environments consist of several stages or workcenters, each with a number of machines in parallel. The machines at a workcenter may be identical or may not be exactly identical (Pinedo, 2002).

Identical Parallel Machines ($\alpha = P$): In this case, m identical machines are in the system and any job can run on any machine, each having the same processing time.

Unrelated Parallel Machines ($\alpha = R$): In this case, m identical machines are in the system and any job can run on any machine, each having the different processing time.

2.2.1.3 Flow Shop

In a flow shop environment ($\alpha = F$), all routes of all jobs are identical and all jobs visit the same machines in the same sequence. The job sequence may vary from machine to machine since jobs may be resequenced between the machines. In the classical flow shop problem, processing times are sequence-independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without preemption (Pinedo, 2002).

In the flow shop, each job needs to visit the machines in the same fixed order, which is assumed to be $\{1, 2, \dots, m\}$. A machine after the completion of current job chooses to process any job in its queue. In many applications, the machines also need to process the jobs in the order that the jobs enter in the queue. Such schedules are called as first in first out (FIFO) schedules and the flow shop is referred to as the permutation flow shop (PFSP).

2.2.1.4 Job Shop

The job shop scheduling problem ($\alpha = J$) is to determine a schedule of jobs that have prespecified operation sequences in a multi-machine environment. In the classical job shop scheduling problem, n jobs are processed for completion on m unrelated machines. For each job, technology constraints specify a complete, distinct routing which is fixed and known in advance. Processing times are sequence-independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without preemption (Pinedo, 2002).

The general JSSP is strongly NP-hard. In order to match today's market requirements, manufacturing systems must become more flexible and efficient. To achieve these objectives, the systems need not only automated and flexible machines, but also flexible scheduling systems. The flexible job shop scheduling problem extends JSSP by assuming that, for each given operation, there is at least one instance of the machine type necessary to perform it.

2.2.1.5 Flexible Flow Shop

Flexible flow shop environment ($\alpha = FF_c$) is a generalization of the flow shop. It consists of a number of stages in series with a number of machines in parallel at each step. The routes of all jobs are identical and all jobs visit the same workcenters in the same sequence. The job sequence may vary from machine to machine since jobs may be resequenced between the machines.

In the flow shop, each job needs to visit the machines in the same fixed order, but in the flexible flow shop, each job needs to visit the any machines which are parallel in the same fixed order. In the classical flexible flow shop problem, processing times are sequence-independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without preemption.

2.2.1.6 Flexible Job Shop

A generalization of the job shop is the flexible job shops with workcenters that have multiple machines in parallel ($\alpha = FJ_c$). The flexible job shop scheduling problem could be formulated as follows. There is a set of n jobs $J = \{J_1, \dots, J_n\}$ to be processed. Each job J_i consists of a predetermined sequence of operations. Each job J_i may have different number of operation. There is a set of m machines $M = \{M_1, \dots, M_m\}$. For each operation $O_{i,j}$, there is a set of alternative machines set $M_{i,j}$ for performing it. Then each job could be processed more than once on the same machine. The processing time of each operation on each machine is predefined. All jobs are released at time 0. All machines are available at time 0, the preemption is not allowed, and each machine could perform at most one operation at any time. There are no precedence constraints among operations of different jobs.

2.2.2 Side Constraints (β)

The side constraints capture various restrictions on the scheduling problem. These are release dates, precedence constraints, setup times, preemption, storage / waiting constraints, machine eligibility, and recirculation.

Release Dates ($\beta = r_j$): The release date r_j of job j is also known as the ready date. It is the time the job arrives at the system, the earliest time at which job j can start its processing.

Precedence Constraints ($\beta = prec$): In scheduling problems, a job often can start only after a given set of other jobs has been completed. Such constraints can be described by a precedence constraints graph.

Setup Times ($\beta = s_{j,k}$): A time period required for preparing a machine. For example, probably the machine needs to be cooled down after the job j before starting the job k . Unless otherwise specified, these set-up times are assumed to be 0.

Preemption ($\beta = prmp$): Sometimes, during the execution of a job, an event forcing the scheduler to interrupt the processing of that job occurs in order to make the machine available for another job.

Storage and waiting constraints ($\beta = block$): In many production systems, the amount of the space available for Work-In-Process (WIP) storage is limited. This puts an upper bound on the number of jobs waiting for a machine. In job shops, this can cause blocking. Suppose the storage space between two successive machines is limited. When the buffer is full, the upstream machine cannot release a job that has been completed into the buffer. Instead, that job has completed its processing and thus prevents that machine from processing another job.

Machine eligibility constraints ($\beta = M_j$): In a parallel machine environment, it may often be the case that the job j can not be assigned to just any of the machines available; it can only go on a machine that belongs to a specific subset M_j . This may occur when the m machines in parallel are not exactly identical.

Recirculation ($\beta = circ$): When some jobs visit a machine group more than once, this is called as recirculation.

2.2.3 Objective (γ)

The objective function decides how the scheduling algorithm is designed. However, there is a large list of possible objective functions depending on the application. The classical objective functions are either of “minsum” or “minmax” type. Oyetunji (Oyetunji, 2009) addressed mathematical expressions for about 29 distinct scheduling objectives.

2.2.3.1 Performance Measures Based on Completion Time

The completion time of the job J_i is the time at which the processing of the job J_i finishes. For a multi operation job, it is the time the last operation of the job J_i finished. Oyetunji’s (Oyetunji, 2009) classification of scheduling criteria based on completion time of jobs are as follows:

The total completion time is the sum of all the completion times of the jobs. A common problem is to minimize the total completion time.

$$\text{Total completion time}(C_{tot}) = \sum_{i=1}^n C_i \quad (2.1)$$

Total weighted completion time is the sum of all the completion times multiplied by relative weights of the jobs. A common problem is to minimize the total weighted completion time.

$$\text{Total weighted completion time } (wC_{tot}) = \sum_{i=1}^n w_i C_i \quad (2.2)$$

Average completion time gives the average time required for completing each job. A common problem is to minimize the average completion time.

$$\text{Average completion time } (C_{avg}) = \frac{1}{n} \sum_{i=1}^n C_i \quad (2.3)$$

Average weighted completion time is the total weighted completion time divided by the number of jobs. Since the number of jobs in any particular problem instance is constant, the total weighted completion time criterion is equivalent to the total weighted completion time criterion. A common problem involves the minimization of the total weighted completion time.

$$\text{Average weighted completion time}(wC_{avg}) = \frac{1}{n} \sum_{i=1}^n w_i C_i \quad (2.4)$$

The maximum completion time also called as makespan is the completion time of the last job. A common problem of interest is to minimize C_{max} , or to minimize the completion time of the last job leaving the system.

$$\text{Maximum completion time } (C_{max}) = \max (C_1, C_2, \dots, C_n) \quad (2.5)$$

2.2.3.2 Performance Measures Based on Flow Time

The flow time of the job J_i is the time that the job J_i spends in the workshop. It is the time interval between the time during which the job is released to the shop and the time during which the processing of the job is completed. Oyetunji's (Oyetunji, 2009) classification of scheduling criteria based on completion time of jobs are as follows:

The total flow time is the sum of all flow times of the jobs. A common problem is to minimize the total flow time.

$$\text{Total flow time}(F_{tot}) = \sum_{i=1}^n F_i = \sum_{i=1}^n (C_i - r_i) = \sum_{i=1}^n C_i - \sum_{i=1}^n r_i \quad (2.6)$$

Total weighted flow time is the sum of all the flow times multiplied by relative weights of the jobs. A common problem is to minimize the total weighted flow time.

$$\text{Total weighted flow time}(WF_{tot}) = \sum_{i=1}^n w_i F_i = \sum_{i=1}^n w_i C_i - \sum_{i=1}^n w_i r_i \quad (2.7)$$

The average flow time gives the average time spent by each job in the shop. A common problem is to minimize the average flow time. The average flow time criterion is equivalent to the total flow time criterion.

$$\text{Average flow time}(F_{avg}) = \frac{1}{n} \sum_{i=1}^n F_i = \frac{1}{n} \sum_{i=1}^n C_i - \frac{1}{n} \sum_{i=1}^n r_i \quad (2.8)$$

Average weighted flow time is considered as an usual problem.

$$\text{Average weighted flow time}(wF_{avg}) = \frac{1}{n} \sum_{i=1}^n w_i F_i = \frac{1}{n} \sum_{i=1}^n w_i C_i - \frac{1}{n} \sum_{i=1}^n w_i r_i \quad (2.9)$$

The maximum flow time is the longest one of the flow times of the jobs. A common problem of interest is to minimize F_{max} .

$$\text{Maximum flow time}(F_{max}) = \max(F_1, F_2, \dots, F_n) \quad (2.10)$$

$$F_{max} = \max\{(C_1 - r_1), (C_2 - r_2), \dots, (C_n - r_n)\} \quad (2.11)$$

2.2.3.3 Performance Measures Based on Lateness

This is the difference between the completion time and the due date which is the expected delivery time of the job. Oyetunji's (Oyetunji, 2009) classification of scheduling criteria based on completion time of jobs are as follows:

The total lateness is the sum of all latenesses of the jobs. A common problem is to minimize the total lateness.

$$\text{Total lateness}(L_{tot}) = \sum_{i=1}^n L_i = \sum_{i=1}^n (C_i - d_i) = \sum_{i=1}^n C_i - \sum_{i=1}^n d_i \quad (2.12)$$

Total weighted lateness is the sum of all latenesses multiplied by relative weights of the jobs. A common problem is to minimize the total weighted lateness.

$$\text{Total weighted lateness}(wL_{tot}) = \sum_{i=1}^n w_i L_i = \frac{1}{n} \sum_{i=1}^n w_i C_i - \frac{1}{n} \sum_{i=1}^n w_i d_i \quad (2.13)$$

The average lateness is total lateness divided by the number of jobs. Therefore, minimizing total lateness also minimizes the average lateness. A common problem is to minimize the average lateness.

$$\text{Average lateness}(L_{avg}) = \frac{1}{n} \sum_{i=1}^n L_i = \frac{1}{n} \sum_{i=1}^n C_i - \frac{1}{n} \sum_{i=1}^n d_i \quad (2.14)$$

The average weighted lateness is total weighted lateness divided by the number of jobs. Also, minimizing total weighted lateness also minimizes the average weighted lateness. A common problem is to minimize the average weighted lateness.

$$\text{Average weighted lateness}(wL_{avg}) = \frac{1}{n} \sum_{i=1}^n w_i L_i = \frac{1}{n} \sum_{i=1}^n w_i C_i - \frac{1}{n} \sum_{i=1}^n w_i d_i \quad (2.15)$$

The maximum lateness (L_{max}) is the longest lateness of the jobs. A common problem of interest is to minimize L_{max} .

$$\text{Maximum lateness}(L_{max}) = \max(L_1, L_2, \dots, L_n) \quad (2.16)$$

$$L_{max} = \max\{(C_1 - d_1), (C_2 - d_2), \dots, (C_n - d_n)\} \quad (2.17)$$

2.2.3.4 Performance Measures Based on Number of Late/Tardy Jobs

A job is said to be late or tardy if it is completed after its due date. Oyetunji's (Oyetunji, 2009) classification of scheduling criteria based on completion time of the jobs are as follows:

$$\text{Let } U_i = \begin{cases} 1, & C_i > d_i \\ 0, & \text{otherwise} \end{cases} \quad i = 1, \dots, n$$

$$\text{Number of tardy jobs (NT)} = \sum_{i=1}^n U_i$$

The number of tardy jobs measures the number of jobs that are completed after their due dates. Typical problem of interest is to minimize the number of tardy jobs. Minimizing number of tardy jobs criterion is equivalent to maximizing number of early jobs criterion.

Average number of tardy jobs (NT_{avg}) = The average number of tardy jobs is the number of tardy jobs divided by the number of jobs. Typical problem of interest is to minimize the average number of tardy jobs. Minimizing the average number of tardy jobs criterion is equivalent to maximizing the average number of early jobs criterion.

2.2.3.5 Performance Measures Based on Tardiness

Tardiness is similar to the lateness except that it carries only positive values. Whenever a job is completed before its due dates, its lateness is negative while its tardiness is zero. Oyetunji's (Oyetunji, 2009) classification of scheduling criteria based on completion time of jobs are as follows:

The total tardiness (T_{tot}) is the sum of all the tardiness of the jobs. A common problem is to minimize the total tardiness.

$$\text{Total tardiness } (T_{tot}) = \sum_{i=1}^n T_i = \sum_{i=1}^n [\max\{0, (C_i - d_i)\}] \quad (2.18)$$

This is the sum of all the tardiness multiplied by the relative weights of the jobs. A common problem is to minimize the total weighted tardiness.

$$\text{Total weighted tardiness } (wT_{tot}) = \sum_{i=1}^n w_i T_i = \sum_{i=1}^n w_i [\max\{0, (C_i - d_i)\}] \quad (2.19)$$

The average tardiness is total tardiness divided by the number of jobs. Therefore, minimizing total tardiness criterion also minimizes the average tardiness criterion. A common problem is to minimize the average tardiness.

$$\text{Average tardiness } (T_{avg}) = \frac{1}{n} \sum_{i=1}^n T_i = \frac{1}{n} \sum_{i=1}^n [\max\{0, (C_i - d_i)\}] \quad (2.20)$$

The average weighted tardiness is the total weighted tardiness divided by the number of the jobs. Minimizing total weighted tardiness also minimizes the average weighted tardiness. A common problem is to minimize the average weighted tardiness.

$$\text{Average weighted tardiness } (wT_{avg}) = \frac{1}{n} \sum_{i=1}^n w_i T_i = \frac{1}{n} \sum_{i=1}^n w_i [\max\{0, (C_i - d_i)\}] \quad (2.21)$$

The maximum tardiness is the longest tardiness of the jobs. A common problem of interest is to minimize T_{\max} .

$$\text{Maximum tardiness } (T_{\max}) = \max (T_1, T_2, \dots, T_n) \quad (2.22)$$

$$T_{\max} = \max\{0, (C_1 - d_1), (C_2 - d_2), \dots, (C_n - d_n)\} \quad (2.23)$$

2.2.3.6 Performance Measures Based on Earliness

Earliness is the opposite of lateness; hence, whenever the lateness is negative, the earliness is positive, and whenever the lateness is positive, the earliness is zero.

Oyetunji's (Oyetunji, 2009) classification of scheduling criteria based on completion time of the jobs are as follows:

The total earliness (E_{tot}) is the sum of all earlinesses of the jobs. A common problem is to maximize the total earliness.

$$Total\ earliness(E_{tot}) = \sum_{i=1}^n E_i = \sum_{i=1}^n [\max\{(d_i - C_i), 0\}] \quad (2.24)$$

Total weighted earliness is the sum of all earliness multiplied by the relative weights of the jobs. A common problem is to maximize the total weighted earliness.

$$Total\ weighted\ earliness\ (wE_{tot}) = \sum_{i=1}^n w_i T_i = \sum_{i=1}^n w_i [\max\{(d_i - C_i), 0\}] \quad (2.25)$$

The average earliness is total earliness divided by the number of jobs. A common problem is to maximize the average earliness. Therefore, maximizing total earliness criterion also maximizes the average earliness criterion.

$$Average\ earliness\ (E_{avg}) = \frac{1}{n} \sum_{i=1}^n T_i = \frac{1}{n} \sum_{i=1}^n [\max\{(d_i - C_i), 0\}] \quad (2.26)$$

The average weighted earliness is the total weighted earliness divided by the number of jobs. A common problem is to maximize the average weighted earliness. Maximizing the total weighted earliness also maximizes the average weighted earliness.

$$Average\ weighted\ earliness\ (wE_{avg}) = \frac{1}{n} \sum_{i=1}^n w_i T_i = \frac{1}{n} \sum_{i=1}^n w_i [\max\{(d_i - C_i), 0\}] \quad (2.27)$$

The maximum earliness (E_{\max}) is the longest earliness of the jobs. A common problem of interest is to maximize E_{\max} .

$$\text{Maximum earliness } (E_{\max}) = \max (E_1, E_2, \dots, E_n) \quad (2.28)$$

$$E_{\max} = \max\{(d_1 - C_1), (d_2 - C_2), \dots, (d_n - C_n), 0\} \quad (2.29)$$

2.3 General Assumption of Scheduling Problems

To simplify the complexity of manufacturing and service environments, most of the scheduling problems are solved under assumptions associated with job characteristics and objective functions.

The following list gives a set of these assumptions and points out when some of them are inadequate to represent a realistic scheduling problem (Pinedo,2002)

Batches of jobs are always treated as a single job: Although this assumption may be appropriate in many situations, in case of large lots, it may produce poor quality schedules. The classical machine scheduling formulation does not allow starting to process on succeeding machine until all the parts in the lot are processed on the preceding machine. However, a better schedule may be obtained by transferring, a part of a batch to the preceding machine before the completion of the whole batch.

Preemption is not allowed: In most scheduling problems, preemption occurs when an operation is stopped and resumed at a later time. For example, a high priority job enters in the system, a machine suspends to process a low priority job and start processing an high priority job. To simplify the complexity, it is assumed that the preemption is not allowed

Each job visits all machines exactly once: In most of the job shops, the preemption may occur when an operation is stopped, due to the reason such as arrival

at a high priority job. For example, if a high priority job enters in the system, a machine suspends to process low priority job and start processing high priority job. To simplify this complexity, it is assumed that preemption is not allowed.

Machines are always available: In practice, machines may not be available because of maintenance and failures. To overcome the uncertainty, machines are assumed to be available all the time.

Job ready times are all known in advance: This is the characteristic that distinguishes deterministic and stochastic problems. In a deterministic environment, all jobs are ready for processing at time zero, and the jobs are ready for processing at different times in stochastic environment.

The problem is purely deterministic: In practice, scheduling problems are stochastic in nature. Machine failures, unpredictable processing times caused the uncertainty, so the scheduling problems are mostly assumed deterministic.

Machines are the only resources modelled: To simplify the complexity of scheduling problems, it is assumed that machines are the only resources that are modelled, but in practice, it may be necessary to model additional resources such as transportation devices, tools or skilled labor.

2.4 Summary

Scheduling problems involve finding optimal schedule under various objectives, different machine environments, and characteristics of the jobs. A detailed description of machine scheduling and objective functions were presented in this chapter. In this thesis, flexible job shop scheduling problem with sequence dependent setup time with completion time based objective is studied.

CHAPTER THREE

FLEXIBLE JOB SHOP SCHEDULING

3.1 Introduction

One of the most necessary subjects in planning and managing processes of manufacturing environments is the scheduling of operations (Pinedo, 2002). The Job-Shop Scheduling Problem (JSSP) is the most complicated and typical problem of all kinds of production scheduling problems (Chen, 2006). A classical job shop, denoted $Jm \parallel C_{max}$ by using the notation presented in Pinedo (Pinedo, 2002), refers to a job shop with a single machine in each workcenter and a makespan (C_{max}) as the scheduling objective. JSSP is one of the hardest combinatorial optimization problems. It belongs to the class of non-deterministic polynomial-time hard (NP-hard) problems, consequently there are no known algorithms guaranteeing to give an optimal solution and run in polynomial time.

JSSP considers a set of jobs to be processed on a set of machines. Each job is defined by an ordered set of operations and each operation is assigned to a machine with a predefined constant processing times. The order of the operations within the jobs and its corresponding machines are fixed a priori and independently from job to job. For solving this problem, we need to find a sequence of operations on each machine respecting some constraints and optimising some objective function. Many different types of objectives are important in manufacturing settings. In practice, the overall objective is often a composite of several basic objectives. The most important ones of these basic objectives are completion time, flow time, lateness, tardiness, and earliness.

Flexible Job-shop Scheduling Problem (FJSSP) which denoted $FJ_c \parallel C_{max}$ is a generalization of the job shop and the parallel machine environment, which provides a closer approximation to a wide range of real manufacturing systems (Cheng, Gen and Tsujimura, 2009).

FJSSP can be decomposed into two sub-problems; assigning the operations to machines (the routing problem) and sequencing the operations on the machines (the sequencing problem) in order to minimize the performance measures. Then, FJSSP becomes more difficult than the classical JSSP because it contains an additional problem such as assigning operations to machines. FJSSP is NP-hard since it is an extension of the JSSP (Gao, Sun and Gen 2008).

3.2 Solution Approaches for Scheduling Problems

There are many more scheduling problems that are intrinsically very hard. These problems are referred to as NP-hard. They are typically combinatorial problems that cannot be formulated as linear programs and there are no simple rules or algorithms that yield optimal solutions in a limited amount of computer time. There are various classes of methods that are useful for obtaining optimal solutions for such NP-hard problems. Solution approaches can be classified as exact solution approaches and heuristic approaches.

Developing an exact method for determining an optimal schedule is one of the current research areas in job shop scheduling. Optimization techniques are capable of finding an optimal solution for scheduling a small number of jobs and machines within a reasonable amount of time. Moreover there are methods that can be applied to these optimization techniques for improving solution speed and solving larger problem sizes. In scheduling more complex job shop problems, optimization techniques may only be able to solve a very small problem. However, developing an optimization technique may still lead to the creation of an approximation algorithm for solving larger problem sizes. A number of researchers develop an optimization technique and then use it in a preliminary test for their proposed heuristic approaches for large-sized problems (Pinedo, 2002).

3.2.1 Exact Optimisation Methods

Exact optimisation methods are classified into two groups. First one is Dynamic Programming which solve complex problems by breaking them down into simpler subproblems. Second one is Integer Programming which is a mathematical program in which some or all of the variables are restricted to be integers.

3.2.1.1 Dynamic Programming

Dynamic Programming can be applied to problems that are solvable in polynomial time as well as problems that are NP-Hard. Dynamic Programming is a method for solving complex problems by breaking them down into simpler steps until it finds a solution for the original problem.

Dynamic programming enumerates in all possible solutions. The problem is divided into a number of stages, and at each stage a decision is required which impacts on the decisions to be made in larger stages (Pinedo, 2002).

3.2.1.2 Integer Programming

When a planning and scheduling problem can be formulated as an Integer Program, the best known approaches are branch and bound, cutting plane and hybrid methods.

The branch and bound method consists of two fundamental procedures: branching and bounding. Branching is the procedure of partitioning a large problem into two or more sub-problems usually and mutually exclusive. Furthermore, the sub-problems can be partitioned in a similar way. Bounding calculates a lower bound on the optimal solution value for each sub-problem generated in the branching process. Cutting plane methods focuses on the linear program relaxation of the integer program. These methods generate additional linear constraints that must be satisfied

so that the variables become integers. Hybrid methods typically combine ideas from various different approaches (Pinedo, 2002).

3.2.2 Heuristic Approaches

Scheduling problems belong to a broad class of combinatorial optimization problems. To solve these problems, one tends to use optimization algorithms which certainly always find optimal solutions. However, not for all optimization problems, polynomial time optimization algorithms can be constructed. This is because of the fact that some of the problems are NP-hard. In such cases one often uses heuristic algorithms which tend towards but do not guarantee the finding of optimal solutions for any instance of an optimization problem (Cheng et al., 2009).

3.2.2.1 Dispatching Rules

A dispatching rule prioritizes all the jobs that are waiting for processing on a machine. Dispatching rules can be classified in various ways. A distinction can be made between static and dynamic rules. Static rules are not time dependent. They are just a function of the job or the machine data. Dynamic rules are time dependent. A second way of classifying dispatching rules is made according to the information they are based upon. A local rule uses only information pertaining to either the queue where the job is waiting or the machine where the job is queued. A global rule may use information pertaining to other machines, such as processing time of the job on the next machine on its route or the current queue length at that machine (Pinedo, 2002).

3.2.2.2 Local Search

Important classes of improvement type algorithms are the local search procedures. A local search procedure does not guarantee an optimal solution. It usually attempts to find a better schedule in the neighbourhood of the current one. Local search algorithms may be run several times on the same problem instance. At each iteration,

a local search procedure performs a search within the neighbourhood and evaluates the various neighbouring solutions.

Local search in the simplest form, the hill-climbing, stops as soon it counters a local optimum. NP-Hard problems often possess many local optima, even this remedy may not be potent enough to yield satisfactory solutions. In view of this difficulty, several extensions of local search have been proposed, which offer the possibility to escape local optima by accepting occasional deteriorations of the objective function. In what follows we discuss successful approaches based on the related ideas, namely simulated annealing and tabu search. These properties are then used to construct a new population which contains a better solution than the previous one. This technique is known as genetic algorithm (Pinedo, 2002).

3.2.2.2.1 Tabu Search (TS). Tabu search uses a local search procedure to iteratively move from a solution to another solution in the neighbourhood, until some stopping criterion has been satisfied. Tabu search for scheduling problems starts from an *initial solution* which can be obtained by various methods such as dispatching rules, insertion methods, and random methods. In a transition from current solution to a new solution a *neighborhood structure* is defined in order to create a subset of possible neighbors to be chosen for performing a *move*. The best neighbor, which is not in the *tabu list* or satisfies the *aspiration criterion* if it is in the tabu list, is selected based on the neighborhood structure, and then the move is performed, which leads to a new solution. The selected neighbor is added into the tabu list which indicates a short-term memory for the recent move history. The aspiration criterion is used when the best neighbor is in tabu list but performing a forbidden move may lead to a better solution. The stopping criterion can be defined in two ways: stop the search when reaching a lower bound, or stop the search after completing a maximum number of iterations (Glover, 1990)

3.2.2.2.2 Genetic Algorithm (GA). The genetic algorithm is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions for optimization and search problems. Genetic algorithms belong to

the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover (Pinedo, 2002).

3.2.2.2.3 Shifting Bottleneck Procedure (SB). The idea of the SB procedure is to schedule each machine in a job shop optimally under the condition that disjunctive arc directions in each optimal schedule of every single machine concur with an optimal job shop schedule. The general steps of SB consist of sub-problem identification and optimization, bottleneck machine determination, sequencing the bottleneck machine, and re-optimization of each scheduled machine. These steps are repeated until all the machines are scheduled. The sub-problem identification and optimization intends to find an optimal sequence of jobs in each unscheduled machine. Each sub-problem consists of a number of operations that are subject to release dates and due dates that are determined by sequences of operations on other machines. The sequence of operations on a given machine is determined by minimizing the maximum lateness in the associated sub-problem. Then the bottleneck machine is the machine which has the highest maximum lateness from previous step. Next, disjunctive arcs with regard to the optimal sequence of jobs in the bottleneck machine are inserted into the disjunctive graph that represents the job shop problem. In the re-optimization step, for each scheduled machine, a job sequence in a particular machine is redefined by finding a new optimal solution of this machine sub-problem, while keeping job sequences fixed in the remaining machines. This re-optimization method is repeated for all scheduled machines (Pinedo, 2002)

3.2.2.2.4 *Simulating Annealing (SA)*. Simulated Annealing procedure is a search process originating from the fields of material science and physics. It is a probabilistic method for finding the global minimum that may possess several local minima. The simulating annealing procedure goes through a number of iterations. The algorithm, in its search for an optimal schedule, moves from one schedule to another. At iteration, a search for a new schedule is conducted within the neighbourhood (Pinedo,2002)

3.3 Literature Review

This thesis addresses the problem of scheduling flexible job shops with sequence dependent setup time with makespan objective ($FJ_c | s_{j,k} | C_{\max}$). Besides makespan; setup times and workload are examined. Genetic algorithm has been applied for solving the problem. In this section, scheduling studies in flexible job shop scheduling problems are focused on. Sequence dependent setup times studies are also included, so this thesis deals with sequence dependent flexible job shop scheduling problems.

Bruker and Schlie (1990) were among the first to address flexible job shop scheduling problem. They developed a polynomial algorithm for solving the flexible job-shop scheduling problem with two jobs.

In order to solve the realistic case with more than two jobs, local search algorithms such as TS, GA, SB, SA are developed.

Brandimarte (1993) is the first researcher who applies TS to flexible job shops scheduling problem with makespan objective ($FJ_c || C_{\max}$) and the first researcher to use the decomposition for the FJSP. He solved the routing sub-problem by using some existing dispatching rules and then focused on the scheduling sub-problem, which is solved by using a tabu search heuristic.

Hurink, Jurisch and Thole (1994) proposed a tabu search heuristic to flexible job shop scheduling problem with makespan objective ($FJ_c \parallel C_{\max}$) in which reassignment and rescheduling are considered as two different types of moves. Initial solutions are calculated by using a fast heuristic based on insertion techniques and beam search. The tabu search algorithms yield excellent results for almost all problems.

Dauzere-Peres and Paulli (1997) defined a new neighbourhood structure for flexible job shop scheduling problem with makespan objective ($FJ_c \parallel C_{\max}$) where there was no distinction between reassigning and re-sequencing an operation, and proposed a TS algorithm.

Bruker and Neyer(1998) proposed the best insertion of an operation in neighbourhood function to solve the FJSP with a makespan objective ($FJ_c \parallel C_{\max}$), but the algorithm they suggest is too time consuming. In order to reduce the computational effort, they propose a faster algorithm that guarantees only the feasibility of an insertion.

Valls, Perez and Quintanilla (1998) presented the application of TS in a generalized job shop with parallel machines, job batches, setup times, and release and due dates ($FJc \mid r_j, s_{j,k}, \text{due-date} \mid C_{\max}$). This problem deals with a makespan minimization by treating the due dates as constraints. The proposed TS have two aspects. First, the core of the heuristic is a tabu threshold algorithm which uses a set of moves aimed at resolving violated constraints. Second, the TS algorithm supplements the first algorithm with two diversification strategies which depends on the state of the search. The overall proposed algorithms seem very complicated but overall performance is superior as being able to obtain known optimal solution over 95% of 190 randomly generated problems in small run time on 66 MHz 486DX2 processor (for instance 67 seconds for 100 jobs–17 machines including parallel machines).

Mastrolilli and Gambardella (2000) improved *Dauzere-Peres'* tabu search techniques and presented two neighbourhood functions. Their TS is the well-known efficient approach for solving FJSP with makespan objective ($FJ_c \parallel C_{\max}$). Their approach is compared with other TS heuristics developed by *Brandimarte (1993)*, *Hurink et al. (1994)*, *Barnes and Chambers (1996)*, *Dauzere, Perez and Paulli (1997)*, and *Brucker and Neyer (1998)*. They carried out experiments on a 266 MHz Pentium over 178 benchmark problems obtained from the other comparative heuristics. The results show that the *Mastrolilli and Gambardella* heuristic is able to find 120 new upper bounds (of the 178 problems) and 77 of them are optimal solutions.

Fattahi, Mehrabad and Jolai (2007) presents a tabu search algorithm that solves the flexible job shop scheduling problem with sequence dependent setup times to minimize the makespan time ($FJ_c | s_{j,k} | C_{\max}$). The proposed tabu search algorithm is composed of two parts. The first part is a procedure that searches for the best sequence of job operations, and the second part is a procedure that finds the best choice of machine alternatives. Results of the algorithm are compared with the optimal solution obtained by using mathematical model. Computational results indicate that the proposed algorithm can produce optimal solutions in a short computational time for small and medium sized problems.

Chen, Ihlow and Lehmann (1999) developed a GA for a flexible job shop scheduling problem to minimize the makespan ($FJ_c \parallel C_{\max}$). In the representation, encoding each individual requires two chromosomes. The first chromosome defines the routing policy and the second chromosome defines the sequence of the operations on each identical parallel machine. In reproduction, single-point and two-point crossovers are applied in the first chromosome. For the second chromosome, two crossovers called as “order-preserving single point crossover” and “order-preserving two-point crossover” are developed. The GA is tested in a SPARC-workstation on three instances. The results show that the instances with 10 jobs, 10 machines and 32 total operations require an average run time of approximately 16 minutes which is considered to be high with respect to the number of operations.

Wang and Brunn (2000) also developed a GA for a flexible job shop scheduling problem to minimize the makespan ($FJ_c \parallel C_{\max}$). A chromosome consists of m sub-chromosomes where m is the number of machines in the shops. Each sub-chromosome, consisting of n genes where n is the number of jobs, represents the job numbers and their corresponding processing routes. A crossover operator called “sequence-extracting crossover” is developed. Computational results show that the GA can find the optimal solution when solving a 6 job–6 machine benchmark problem within about two minutes running time. However, it is unclear that how the GA performs in larger size problems with identical parallel machines in work centers.

Kacem, Hammadi, and Borne (2003) proposed a genetic algorithm controlled by the assigned model which is generated by the Approach of Localization (AL) to solve FJSP with makespan and workload objectives ($FJ_c \parallel C_{\max}, workload$). Chromosome representation combines both routing and sequencing information. Dispatching rules are then applied for sequencing the operations. Once this initial population is found, they apply crossover and mutation operators to jointly modify assignments and sequences, producing better individual ones.

Pezzella, Morganti and Ciaschetti (2008) adopted many of the choices of *Kacem et al. (2003)* and presented a genetic algorithm (GA) in which uses a mix of different strategies for generating the initial population, selecting individuals. The initial population is generated according to Approach by Localization. This approach takes into account both the processing times and the workload of the machines, i.e., the sum of the processing times of the operations assigned to each machine. The procedure consists of finding the machine with the minimum processing time, fixing that assignment, and then adding this time to every subsequent entry in the same column (machine workload update). Since this approach is strongly dependent on the order in which operations and machines are given in the table, *Pezzella et al.* slightly modify it in two ways: The first one is to search for the global minimum in the processing time table. The second one is to permute the jobs and the machines randomly in the table.

Gao, Gen and Sun (2008) studied the FJSP with a multi-objective approach. They developed a hybrid genetic algorithm (hGA) for this problem with makespan and workload objective ($FJ_c \parallel C_{\max}, workload$). Their algorithm is the well-known competitive genetic algorithm for solving the FJSP.

Gen, Gao and Lin (2009) developed a new approach hybridizing genetic algorithm with shifting bottleneck to fully exploit the global search of genetic algorithm and the local search of shifting bottleneck for solving multi-objective flexible job shop scheduling problem with makespan and workload objective ($FJ_c \parallel C_{\max}, workload$). The genetic algorithm uses two vectors to represent each solution candidate of the FJSP. Phenotype-based crossover and mutation operators are proposed to adapt to the special chromosome structures and the characteristics of the problem. The shifting bottleneck works over two kinds of effective neighbourhood, which use interchange of operation sequences and assignment of new machines for operations on the critical path. In order to strengthen the search ability, the neighbourhood structure can be dynamically adjusted in the local search procedure. The performance of the proposed method is analyzed by numerical experiments on three representative problems.

Defersha and Chen (2009) modified Kacem's approach by taking into account setup times and they proposed a parallel genetic algorithm to solve FJSP with sequence dependent setup time ($FJ_c \mid s_{j,k}, batch \mid C_{\max}$). They developed a comprehensive model for FJSP by considering several factors in an integrated manner and assume that the jobs have sequence dependent setup time. The number of researches considering this issue in FJSP is limited. The model also allows a given setup to be either attached or detached depending on the actual requirements. Other important factors incorporated in the proposed model are machine release date. The last incorporated factor is the concept of time lag. In order to solve this NP-hard problem efficiently, they proposed a parallel genetic algorithm (PGA) implemented by using "island" model. In this parallelization model, subpopulations are separately

evolved in several processors and this subpopulations exchange individuals periodically. The PGA was executed on a high-performance computing environment composed of multiple interconnected workstations. The developed model and solution procedure were extensively tested with medium and large problem instances. The results obtained by using the PGA are very promising.

Fattahi and Fallahi (2010) used Kacem's approach under dynamic flexible job shop conditions ($FJ_c \parallel C_{\max}, workload$). In this research, a mathematical model for the dynamic flexible job shop scheduling problem and a meta-heuristic algorithm based on GA is developed. The proposed algorithm improves the efficiency and stability of schedules. Numerical experiments were used for evaluating the performance and the effectiveness of the proposed algorithm. It is concluded that the bi-objective model can improve the efficiency and stability of schedules based on a real example from a part making industry. The experimental results show that the proposed algorithm is capable of producing the optimal solutions for small sized problems. Also the results of the proposed algorithm are between the upper and the lower bounds for medium sized problems. These upper and lower bounds of are computed by the branch and bound method. So, the proposed algorithm is capable of reaching near optimal solutions for medium sized problems. The standard deviation of the solutions are equal to zero for small sized problems which show the high quality of the algorithm on small sized problems. Nevertheless, the standard deviation of the solutions is increased for the medium and large sized problems. So the convergence of the proposed algorithm is decreased for large sized problems. Also the genetic algorithm parameters are adjusted based on preliminary experiments.

Mason, Fowler and Carlyle (2002) proposed a modified shifting bottleneck called MSB for minimizing total weighted tardiness in complex job shops. ($FJ_c \parallel \text{batch} \mid \text{total weighted tardiness}$) They defined the complex job shop as a flexible job shop with sequence-dependent setup times, recirculation, batching, and possibly non-zero ready times. This type of production environment can be found in wafer fabrication facilities in semiconductor industries. In the proposed heuristic, a disjunctive graph is

used for representing the problem. The Apparent Tardiness Cost with Setups (ATCS) index for parallel machines scheduling problem, developed by *Lee and Pinedo (1997)*, is modified to accommodate batching problems. The modified ATCS, called BATCS, is used in the subproblem solution procedures. The re-optimization step, considering the newly added disjunctive arcs for the selected machine, is the optional step in MSB. To assess the ability of their proposed models, a benchmark problem is used for comparing the MSB with a number of existing dispatching rules. The results show that the MSB needs the re-optimization step in order to obtain a better solution but the computational time will increase about 50%.

Mason and Oey (2003) reported that cyclic paths in a disjunctive graph may occur when scheduling complex job shops, resulting in feasible schedules during the execution of the MSB heuristic (*Mason et al. 2002*). To eliminate the cyclic paths, they modified the disjunctive graph representing the complex job shop problems and identified all possible causes of infeasible schedule generation. Then, the Cycle Elimination Procedure (CEP) was developed to tackle this problem. The mechanism of CEP is spotting batching nodes that cause the cyclic schedule, and then removing them. Their proposed procedure is applied in practical wafer fabrication. The results indicated that solution times increased about 10% with a satisfied level of solution quality. However, the algorithm executed for only two product types and for one shift. So utilization of the proposed MSB with CEP is still unclear.

Ho, Tay and Lai(2007) proposed an architecture for learning and evolving of Flexible Job-Shop schedules called LEarnable Genetic Architecture (LEGA) with makespan objective ($FJ_c \parallel C_{\max}$). LEGA provides an effective integration between evolution and learning within a random search process unlike the “canonical evolution algorithm”, where random elitist selection and mutational genetics are assumed; through LEGA, the knowledge extracted from previous generation by its schemata learning module is used for influencing the diversities and the qualities of offsprings. In addition, the architecture specifies a population generator module that generates the initial population of schedules and also trains the schemata learning module. A large range of benchmark data taken from literature and generated by

these researchers are used for analyzing the efficacy of LEGA. Experimental results indicated that an instantiation of LEGA called GENACE outperforms current approaches using canonical EAs in computational time and quality of schedules.

Tay and Ho (2004) developed new algorithm called as GENACE to solve FJSSP with makespan and workload objective ($FJ_c \parallel C_{\max}, workload$). We show how composite dispatching rules (CDRs) are used for solving the FJSP with recirculation by themselves and for providing a bootstrapping mechanism to initialize GENACE. They then adopted a cultural evolutionary architecture to maintain the knowledge of schemata and resource allocations learned over each generation. The belief spaces influence mutation and selection over a feasible chromosome representation. Experimental results show that GENACE obtains better upper bounds for 11 out of 13 benchmark problems, with improvement factors of 2 to 48 percent when compared to results by *Kacem et al, Brandimarte*.

Xia and Wu (2005) made use of particle swarm optimization (PSO) to assign operations on machines and SA algorithm to schedule operations for JSSP with makespan objective ($FJ_c \parallel C_{\max}$).

Rossi and Dini (2007) proposed an ant colony optimisation-based software system for solving FMS scheduling in a job-shop environment with routing flexibility, sequence-dependent setup and transportation time with makespan criterion ($FJ_c |s_{i,j}, block| C_{\max}$). The objective in this thesis is to develop several procedures for scheduling flexible job shops with sequence dependent setup time with makespan objective. In literature, minimizing makespan is usually found in job shop scheduling problems because it concerns the reduction of the completion time for only the last complete job by moving operations in the longest path in a given schedule. This thesis takes into account flexible job shop problems including non-identical parallel machines, sequence-dependent setup times. These characteristics usually occur in real world production environments. In the last decade, processing restrictions, such as sequence-dependent setup times, recirculation, batch non-zero ready times as well

as due-date based objectives in flexible job shop scheduling problems, have received more attention from many researchers. In this research zero ready times, makespan objectives and sequence dependent setup time are used.

Table 3.1 Some applications of flexible job shop scheduling problems

Problem class	Algorithm	Citation
flexible job shop makespan two jobs	Polynomial Algorithm	- Bruker and Schlie(1990)
flexible job shop makespan	TS	- Brandimarte(1993) - Hurink, Jurisch, and Thole (1994). - Barnes and Chambers (1996) - Dauzere-Perez and Paulli (1997) - Brucker and Neyer (1998) - Tung, and Nagi(1999) - Chen et al. (1999) - Wang and Brunn (2000) - Mastrolilli and Gambardella (2000)
flexible job shop makespan	GA	- Chen et al. (1999) - Wang and Brunn (2000) - M. Zandieh,I. Mahdavi,A. Bagheri(2008)
flexible job shop sequence dependent setup time makespan	TS	- Mehraba and Fattahi(2007)
flexible job shop multiobjective	Hybrid GA	- Gao,Gen M, Sun L, Zhao X(2007) - Mitsuo Gen, Jie Gao(2009)
flexible job shop sequence dependent setup time makespan	ACO	- Andrea Rossi, Gino Dini(2007)

Table 3.1 (Cont) Some applications of flexible job shop scheduling problems

Problem class	Algorithm	Citation
flexible job shop multiobjective	GA+Approach by localization	- Kacem, Hammadi, and Borne(2003) - M. Gen (2005) - F.Pezella, G.Morganti,G. Ciaschetti (2007)
flexible job shop ready time, sequence-dependent setup time, reentrant job, batch multiobjective	Paralel GA+ Approach by localization	- Defersha and Chen(2009)
Dynamic flexible job shop makespan	GA+Approach by localization	- Fattahi and Fallahi(2010)
flexible job shop ready time, sequence-dependent setup time, reentrant job, batch total weighted tardiness	SB	- Mason et al. (2002) - Mason and Oey (2003)
flexible job shop sequence dependent setup time makespan	Matemactical model+heuristics	- Parviz Fattahi, Mohammed Saidi Mehrabad(2006) - R. Moghaddas, M.Houshmand (2008)
flexible job shop multiobjective	PSO+SA	-Xia and Wu(2005)
flexible job shop total tardiness	TS	-Valls et al. (1998)

3.4 Summary

Flexible job shop scheduling problems have been widely presented in the literature. GA and TS are reasonable approaches for scheduling problems since they have been applied for scheduling several job shop classes and obtained fairly good solutions. In flexible job shop scheduling problems, infeasible schedules may be generated during searching. Prevention of infeasible schedules is a key characteristic in the development of an effective metaheuristic. In GAs, good encodings that lead to feasible schedules are required.

CHAPTER FOUR

GENETIC ALGORITHMS

4.1 Introduction

A genetic algorithm (GA) is a class of adaptive stochastic optimization algorithms involving the search and the optimization. Genetic algorithms were first used by John Holland (1975) for the formal investigation of the mechanisms of natural adaptation, but the algorithms have been modified to solve computational search problems. Modern GA deviates greatly from the original form proposed by Holland. There is no single firm definition for a genetic algorithm.

GAs are successfully used for finding solution to a range of the optimization problems. GA solves a large scale of design, control, scheduling or other engineering optimization problems. The general idea is to start with randomly generated solutions, to implement an evolutionary process, and then to search for better solutions while eliminating poor solutions from the current generation to the next generation.

4.2 Solving Scheduling Problems with Genetic Algorithms

Most scheduling problems are NP-Hard, the time required for optimally solving the problem increases exponentially with the size of the problem. GA are successfully used for solving the scheduling problems and well suited to solve production scheduling problems.

Genetic algorithms operate on a population of solutions rather than a single solution. In production scheduling, this population of solutions consists of many answers that may have different, sometimes conflicting objectives. To apply a genetic algorithm to a scheduling problem, we must first represent it as a genome. One way to represent a scheduling genome is to define a sequence of tasks. Each task and its corresponding start time represent a gene.

There are three basic approaches of applying the genetic algorithms to a scheduling problem (Gen et al., 1999):

1. Adapt problems to the genetic algorithms
2. Adapt the genetic algorithms to problems
3. Adapt both the genetic algorithms and problems

In this Chapter, we focus on the representation of solutions and design of genetic operators for solving the flexible job shop scheduling problem.

4.3 Genetic Algorithm

GA is a search method that mimics the process of natural evolution. This heuristic is routinely used for generating useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Each individual is characterized by its fitness. The fitness of an individual is measured by the associated value of the objective function. The procedure works iteratively, and each iteration is referred to as a generation. At each iterative step, a number of different solutions are generated and carried over to the next generation. And this procedure is repeated until some stopping criterion is met.

A population of individuals is maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables. To continue the genetic analogy, these individuals are likened to chromosomes and the variables are analogous to the genes. Thus a chromosome is composed of several genes. A fitness score is assigned to each solution representing the abilities of an individual to compete. The individual with the optimal or generally near optimal fitness score is sought. The GA aims at using selective breeding of the solutions to produce offspring better than the parents

by combining information from the chromosomes. The GA maintains a population of n chromosomes with associated fitness values. Parents are selected to mate, on the basis of their fitnesses for producing offspring. Consequently, better solutions give more opportunities to reproduce so that offspring inherits characteristics from each parent. New generations of solutions are produced containing, on average, better genes than a typical solution in a previous generation. Each successive generation will contain better partial solutions than previous generations (Shopova and Vaklieva, 2006). Figure 4.1 shows the Architecture of GA.

4.3.1 Chromosome Representation

Genetic representation is a way of representing solutions in evolutionary computation methods. *Mesghouni (1997)* used “parallel machine representation” and “parallel job representation”. *Chen et al. (1999)* divided the chromosome into two parts as A string and B string. A string contains a list of all operations of all jobs and machines selected for corresponding operations, while B string contains a list of operations that are processed on each machine. These representations must be adjusted for feasibility. *Tay and Ho (2007)* used the chromosome representation having two parts, namely “operation order” which was adopted from *Ramiro (2003)* and “machine selection” which use an array of binary values to present machine selection.

Kacem et al. (2003) used a type of representation called as “machine assignment representation”. An array consisting of binary values are used for presenting the machine selection. The best reported performance results so far belong to Kacem. Each unit value in the assignment table maps a machine to a corresponding operation. After using an assignment algorithm to find a set of feasible schedules, they are used as an initial population for GA.

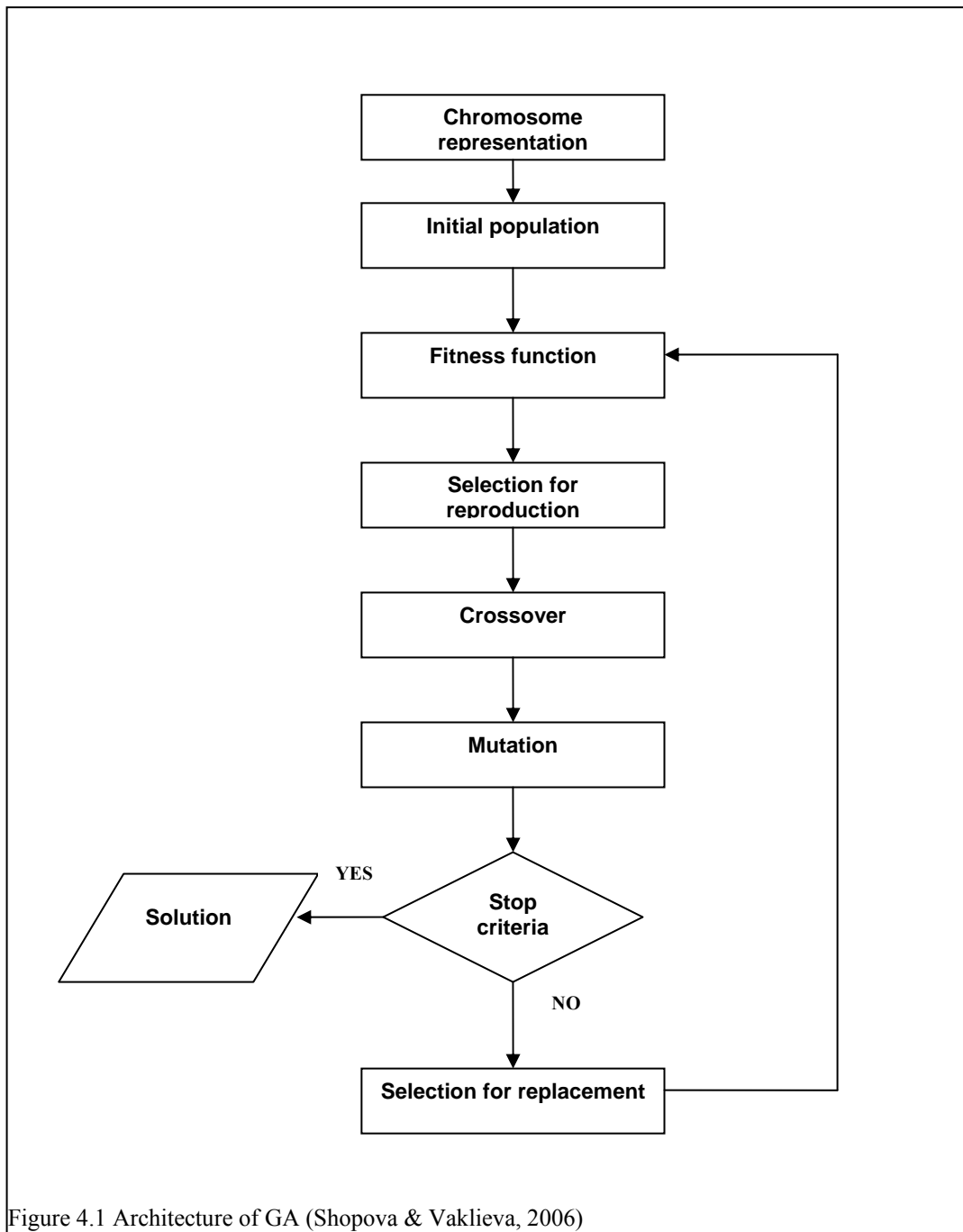


Figure 4.1 Architecture of GA (Shopova & Vaklieva, 2006)

Several different chromosome representations have been proposed for the JSP. Gen et al. (1999) summarized chromosome representations for the job shop scheduling problem.

operation-based representation: This representation uses a permutation with m -repetitions of job numbers for the problem with n jobs and m machines. A job is represented a set of operations that has to be scheduled on m machines. Each job occurs m times in the permutation. By scanning the permutation from left to right, the k^{th} occurrence of a job number refers to the k^{th} operation in the technological sequence of this job. No infeasible solutions will result in this representation (Gen et al.,1999).

job-based representation: This representation merely encode the chromosome according to the job sequence. Each job occurs one time. By scanning the encoding from left to right, all the operations of the first job in the representation scheduled first (Gen et al.,1999).

preference list-based representation: In this representation, the chromosome represents the preferences of each job. There are M sub-chromosomes in each chromosome where each sub-chromosome represent the preference of the jobs on that machine (Gen et al.,1999).

job pair relation-based representation: In this representation, a chromosome is symbolized by a binary string, where each bit stands for the order of a job pair for a particular machine (Gen et al.,1999).

priority rule-based representation: This representation encodes a chromosome as a sequence of dispatching rules. The job schedule is created by using a heuristic based on the dispatching rules sequence (Gen et al.,1999).

disjunctive graph-based representation: This representation encodes the chromosome as a binary string that corresponds to an ordered list of disjunctive arcs

connecting the different operations to be processed by the same machine (Gen et al.,1999).

completion time-based representation: This chromosome representation is an ordered list of completion times of operations. This representation may yield an infeasible schedule. So,when the crossover operator is applied, special crossover operators should be designed for it (Gen et al.,1999).

machine-based representation: This representation encodes the chromosomes as a sequence of machines. This representation may yield an infeasible schedule. So, when the crossover operator is applied, special crossover operators should be designed for it (Gen et al.,1999).

random keys representation: This representation encodes a solution with random numbers. These values are used as sort keys to decode the solution. Each gene consists of two parts. The integer part of any random key is interpreted as the machine assignment for that job. Sorting the fractional parts provides the job sequence on each machine (Gen et al.,1999).

Among the nine encoding methods, the job-based encoding and the machine-based encoding are the pure literal string; the operation-based encoding, the preference list-based encoding and the priority rule-based encoding are the general literal string. Note that the preference list-based encoding is well used in many studies. It consists of several substrings and one substring is a pure literal string corresponding to an operation sequence to a machine. In this thesis, operation-based representation is used.

4.3.2 Genetic Operators

The performance of the genetic algorithm depends on the choice of good genetic operators. Genetic operator is an operator used in genetic algorithms to maintain genetic diversity. These operators are selection, crossover and mutation operators.

Selection is a genetic operator that chooses a chromosome from the current generation's population for inclusion in the next generation's population. Before making selection into the next generation's population, selected chromosomes may undergo crossover or mutation depending upon the probability of crossover and mutation. Crossover is a genetic operator that combines two chromosomes called parents to produce a new chromosome called offspring. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Mutation is a genetic operator that alters one or more genes in a chromosome from its initial state. This can result in entirely new genes being added to the gene pool. With these new genes, the genetic algorithm may be able to arrive to a better solution. Mutation is an important part of the genetic search to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability (Shopova & Vaklieva, 2006).

4.3.2.1 Selection Operator

Selection is the first operator. There are two types of selection operators applied in genetic algorithms. The first one is selection of reproduction which determines the sampling that will produce the offspring. The second one is the selection of replacement which selects individuals that will be included in the next population.

The selection of reproduction involves randomly choosing members of the population to enter a mating pool. GA includes three biased selection schemes for reproduction; rank based, roulette-wheel, and tournament selection. After the production of new individuals, it is necessary to decide which individuals survive in the next generation. Selection for replacement creates a new generation from the current one and the obtained offspring. Selection of replacement methods replace all parents by their children, only if their children are better fitness values. In this thesis roulette-wheel selection is used.

4.3.2.1.1 Roulette-Wheel Selection. The simplest proportionate selection scheme is roulette-wheel selection, also called as stochastic sampling with replacement. The individuals are mapped to contiguous segments of a line in such a manner that each individual's segment will be equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained. This technique is analogous to a roulette wheel with each slice proportional in size to the fitness (Shopova and Vaklieva, 2006).

4.3.2.1.2 Rank Based Selection. Rank-based selection is similar to the proportionate selection schemes. In it, individual's rank, instead of the fitness function, is used for calculating the selection probability. The latter gives a better chance of chromosomes with small fitness values to take part in the reproductive process and preserves populations from a premature convergence (Shopova and Vaklieva, 2006).

“Linear ranking” and “Square ranking” are two methods which are common in use. In linear ranking, the selection probability is proportional to the rank of each individual. In square ranking, the selection probability is the square of its rank:

4.3.2.1.3 Tournament Selection. In tournament selection, a number of individuals in a tour is randomly chosen from the population and the best individual from this group is selected as parent. This process is repeated as often as individuals must be chosen. The parameter for tournament selection is the tournament size “Tour”. “Tour” takes values ranging from 2 to number of individuals in population (Shopova and Vaklieva, 2006).

4.3.2.2 Crossover Operators

Crossover is the second genetic operator. Various crossover operators have been proposed for literal permutation encodings, such as partial-mapped crossover (PMX), order crossover (OX), cycle crossover (CX), position-based crossover, order-based crossover, etc. (Gen et al.,1999). In this thesis, position based crossover is used.

There are two different basic considerations for designing crossover operators for literal permutation encodings;

1. To make less change when crossing over so as to inherit parents' features as much as possible.
2. To make more change when crossing over so as to explore new patterns of permutation and thereby enhance the search ability.

4.3.2.2.1 *Partial-Mapped Crossover (PMX)*. PMX has the following major steps:

1. Select two cut-points along the string at random. The substrings defined by the two cut-points are called the mapping sections.
2. Exchange two substrings between parents to produce proto-child.
3. Determine the mapping relationship between two mapping sections.
4. Legalize offspring with the mapping relationship (Gen et al.,1999).

4.3.2.2.2 *Order Crossover (OX)*. Order crossover (OX) can be viewed as a kind of variation of PMX that uses a different repairing procedure. OX has the following major steps:

1. Select a substring from one parent at random.
2. Produce a proto-child by copying the substrings into the corresponding positions as they are in the parent.
3. Delete all the symbols from the second parent, which are already in the substring. The resultant sequence contains the symbols the proto-child needs.

4. Place the symbols into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce an offspring (Gen et al.,1999).

4.3.2.2.3 Position-Based Crossover. Position-based crossover firstly generates a random mask and then exchanges relative genes between parents according to the mask. A crossover mask is simply a binary string with the same size of chromosome. The parity of each bit in the mask determines, for each corresponding bit in an offspring, from which the parent will receive that bit. Because uniform crossover will produce illegal offspring for literal permutation encodings, position-based crossover uses a repairing procedure to resolve the illegitimacy. Position-based crossover has the following major steps:

1. Select a set of positions from one parent at random.
2. Produce a proto-child by copying the symbols on these positions into the corresponding positions of the proto-child.
3. Delete the symbols which are already selected from the second parent. The resultant sequence contains only the symbols the proto-child needs.
4. Place the symbols into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce one offspring (Gen et al.,1999).

4.3.2.2.4 Order-Based Crossover. Order-based crossover is a slight variation of position-based crossover in that the order of symbols in the selected position in one parent is imposed on the corresponding ones in the other parent (Gen et al.,1999).

4.3.2.2.5 Cycle Crossover (CX). Cycle crossover (CX) is the same as the position-based crossover. It takes some symbols from one parent and the remaining symbols from the other parent. The difference is that the symbols from the first parent are not selected randomly and only those symbols are selected which defined a cycle according to the corresponding positions between parents. CX works as follows:

1. Find the cycle which is defined by the corresponding positions of symbols between parents.
2. Copy the symbols in the cycle to a child with the corresponding positions of one parent.
3. Determine the remaining symbols for the child by deleting those symbols which are already in the cycle from the other parent.
4. Fill the child with the remaining symbols (Gen et al.,1999).

4.3.2.2.6 Linear Order Crossover (LOX). Order crossover tends to transmit the relative positions of the genes rather than the absolute ones. In the order of crossover, the chromosome is considered to be circular since the operator is devised for the travelling salesman problem. In the job-shop problem, the chromosome can not be considered as circular. For this reason they developed a variant of the OX called Linear Order Crossover (LOX), where the chromosome is considered linear instead of circular. The LOX works as follows:

1. Select sublists from parents randomly.
2. Remove sublist2 from parent 1 leaving some 'holes' (marked with h) and then slide the holes from the extremities towards the center until they reach the cross section. Similarly, remove sublist1 from parent 2 and slide holes to cross section.
3. Insert sublist1 into the holes of parent 2 to form the offspring 1 and insert sublist2 into the holes of parent 1 to form an offspring 2 (Gen et al.,1999).

The crossover operator can preserve both the relative positions between genes and the absolute positions relative to the extremities of parents as much as possible. The extremities correspond to high and low priority operations.

4.3.2.2.7 Subsequence Exchange Crossover. A job sequence matrix is used as encodings. For a n job m machine problem, the encoding is an $m \times n$ matrix where each row specifies an operation sequence for each machine. A subsequence is defined as a set of jobs which are processed consecutively on a machine for both

parents but not necessarily in the same order. This method includes the following two steps:

1. Identify subsequences one for one machine for the parents.
2. Exchange these subsequences machine by machine among parents to create offspring (Gen et al.,1999).

Because it is difficult to maintain the precedence relation among operations in either initial population or offspring by use of the job sequence matrix encoding, Giffler and Thompson algorithm is used to carefully adjust job orders on each machine to resolve the infeasibility and to convert offspring into active schedules.

4.3.2.2.8 Job-Based Order Crossover. The job-based order crossover is designed for the encoding of job-sequence matrix. It has the following steps:

1. Identify the sets of jobs from parents, one set for one machine.
2. Copy the selected jobs of the first parent onto the corresponding positions of the first child machine by machine. Do the same thing for the second child.
3. Full the unfixed position of the first child by the not-selected jobs from left to right according to the order as they appear in the second parent. Do the same thing for the second child (Gen et al.,1999).

4.3.2.2.9 Partial Schedule Exchange Crossover. Gen et al. (1994) proposed a partial schedule exchange crossover for an operation-based encoding. They consider partial schedules as the natural building blocks and intend to use such crossover to maintain building blocks in the offspring. It has the following steps:

1. Identify a partial schedule in one parent randomly and in the other parent accordingly.
2. Exchange the partial schedules to generate proto-offspring.
3. Determine the missed and exceeded genes for the proto-offspring.

4. Legalize offspring by deleting exceeded genes and adding missed genes (Gen et al.,1999).

The partial schedule is identified with the same job in the head and tail of the partial schedule.

4.3.2.2.10 Substring Exchange Crossover. Cheng et al. (1997) gave an another version of partial schedule exchange crossover, called substring exchange crossover. It can be viewed as a kind of adaptation of two cut-points crossover for general literal string encodings. It has the following steps:

1. First, select two cut-points along the string at random. Exchange two substrings defined by the two cuts between two parents to produce proto-children.
2. Determine the missed and exceeded genes for each proto-child by making a comparison between two substrings.
3. Legalize the proto-children by replacing the exceeded genes with the missed genes in a random way (Gen et al.,1999).

4.3.2.3 Mutation Operator

Mutation is the third genetic operator. It is the primary search operator. Mutation operator assure the diversity of the population to prevent the premature convergence of GA. Mutation changes the value of individual genes at random with a certain probability and assures that all the points in the search space are likely to be examined. Several mutation operators have been proposed, such as inversion, insertion, displacement, reciprocal exchange mutation, and shift mutation (Gen et al, 1999). In this thesis insertion mutation and modified version of insertion mutation methods are used.

- ***Inversion mutation:*** Selects two positions within a chromosome at random and then inverts the substring between these two positions.

- ***Insertion mutation:*** Selects a gene at random and inserts it in a random position.
- ***Displacement mutation:*** Selects a substring at random and inserts it in a random position. Insertion can be viewed as a special case of displacement in where substring just contains one gene.
- ***Reciprocal exchange mutation:*** Selects two positions at random and then swaps the genes on these positions. Shift mutation first chooses a gene randomly and then shifts it to a random position of right or left from the gene's position.
- ***Shift mutation:*** Chooses a gene randomly and then shifts it to a random position of right or left from the gene's position.

4.3.3 Genetic Parameters

There may be many parameters which are considered in implementing the GA procedure. Three of the most important parameters are population size, probability of crossover and probability of mutation.

- ***Population size:*** Population size affects the efficiency of the algorithm. If we have smaller population, it would only cover a small search space and may result in poor performance. A larger population would cover more space and prevent premature convergence to local solutions. At the same time, a larger population needs more number of evaluations per generation and may slow down the convergence rate.
- ***Probability of Crossover:*** Probability of crossover or crossover rate is the parameter that affects the rate at which the crossover operator is applied. A higher crossover rate introduces new strings more quickly into the population. A low crossover rate may cause stagnation due to the lower exploration rate.

- ***Probability of Mutation:*** Probability of mutation or mutation rate is the probability of changing the bit position of each string in the new population. A low mutation rate helps to prevent any bit positions from getting stuck to single value, whereas a high mutation rate results in essentially random search.

4.4 Summary

Genetic algorithm is adaptive method which may be used to solve search and optimisation problems. They are based on the genetic processes of biological organisms. Scheduling of large-scale problems includes a number of difficulties for search and optimization techniques. Genetic algorithms are well suited to such problems owing to their adaptability and their effectiveness at searching large spaces. In the following chapter, the genetic algorithm is applied to flexible job shop scheduling problem.

CHAPTER FIVE
SOLVING FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH
SEQUENCE DEPENDENT SET UP TIME BY USING GENETIC
ALGORITHM

5.1 Introduction

This chapter presents a solution approach to a flexible job shop scheduling problem with sequence dependent setup time by using GA. The objective is to minimize the makespan. Besides makespan, workload and setup times are also analysed. In this thesis, we focus on the methods which assign the jobs to the machines, sequence the jobs on the machine, selection methods and mutation methods. Genetic algorithm determines a solution approach to minimize the performance measures by the combination of these methods. Matlab is used for solving this problem.

FJSSP is a generalization of the job shop with workcenters that have multiple machines in parallel. Flexible Job Shop Problems are divided into two subproblems. In the first subproblem, assignments of operations to the machines are made. In the second subproblem, the sequencing problem is solved.

For the first subproblem, assignment methods were used in this thesis. The first method is “Approach by Localization” which assigns each operation to the suitable machine by taking into account the processing times and the workloads of machines. This assignment procedure assumes that there are no setup times. Approach by Localization method was first developed by Kacem (2002). Pazella(2007) modified it by mixing machine and operation places. Chen(2009) resolved it by parallel genetic algorithm (PGA) through the inclusion of sequence dependent setup time. Fattahi(2010) applied this method for solving the dynamic flexible job shop problem scheduling.

The second method is a modified version of the first method. In this method, we modified the assignment procedure by using sequence dependent setup times in addition to processing times and workloads of machines. In the third method, assignments were made randomly just for the comparison purposes.

For the second subproblem, dispatching rules such as shortest processing time (SPT), most work remaining (MWR), longest processing time (LPT), shortest setup time first rule (SSTFR), random rule and neighborhood search were applied for sequencing the machines.

Setup times are not generally added to flexible job shop scheduling problem in the literature due to the complexity of the problem. In this thesis, sequence-dependent setup times are taken into consideration in machine assignments realized not only during the formation of the initial population but also in the calculation of the makespan. In the literature, solution methods of flexible job shop scheduling problems are divided into two as optimum and approximate solutions. Algorithms giving the optimum solution are suitable for small sized problems. Heuristic algorithms are used for large sized problems because finding the optimum solution takes a long time. In this thesis, Genetic Algorithm Method is used among heuristic methods.

5.2 Description of the Problem

This thesis takes into consideration flexible job shop problems including identical or nonidentical parallel machines and sequence-dependent setup times. JSSP considers a set of jobs to be processed on a set of machines. Each job is defined by an ordered set of operations and each operation is assigned to a machine with a predefined constant processing times. The order of the operations within the jobs and its corresponding machines are fixed a priori and independently from job to job. For solving this problem, we need to find a sequence of operations on each machine respecting some constraints and optimising some objective function.

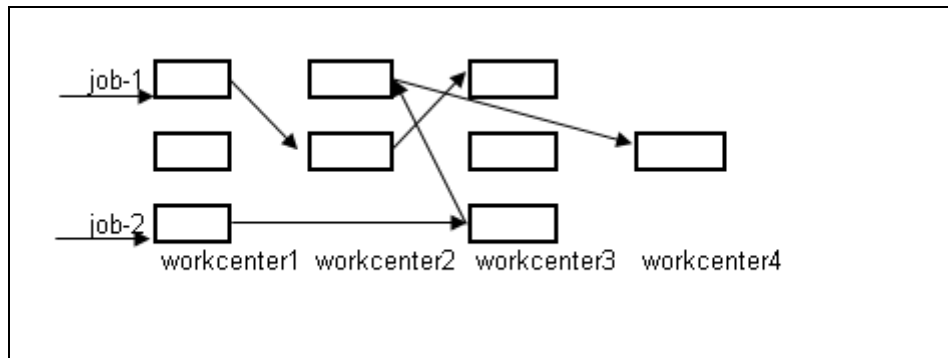


Figure 5.1 Flexible Job Shop Scheduling

Following assumptions are considered for the problem;

1. Each job consists of one fixed sequence of operations.
2. Each machine can process at most one job at a time, and each job can only be processed on one machine at a time.
3. There are no interruptions and cancellations between the jobs.
4. All machines are available continuously.
5. There are identical or nonidentical parallel machines in the system.
6. All jobs can be started at $t = 0$.
7. All machines are available at $t = 0$.
8. There are sequence dependent setup time between the jobs.
9. Setup for a job cannot begin until the job is available to the current work center and the desired machine in the work center is idle.
10. All data including processing times and setup times are known deterministically.
11. There are no precedence constraints among operations of different jobs.
12. Neither release times nor due dates are specified.
13. There are two cases, the first is that all machine can process all the jobs (total flexibility), the second is that all machines can not process all the jobs (partial flexibility).

5.3 Performance Measures

Different types of objectives may be considered in scheduling problems. These are performance measures based on completion time, flow time, lateness, number of late/tardy jobs, earliness, and number of early jobs. In this thesis, one of the completion time based performance measures, makespan was selected, workload and setup times were also evaluated.

Makespan is the difference between the start and finish of a sequence of jobs. The maximum completion time is also called as the makespan which is the completion time of the last job. A common problem of interest is to minimize C_{\max} , or to minimize the completion time of the last job leaving the system. This criterion is usually used for measuring the level of utilization of the machine. The workload is the sum of processing times of the machines. The setup time is the time period required for preparing a machine to be ready for accepting a job.

5.4 Proposed Genetic Algorithm

In FJSP, two subproblems should be solved as it was stated in Chapter Two. In the first subproblem, assignments of operations to the machines are made by using assignment methods. In the second subproblem, sequencing problem is solved by dispatching rules and neighborhood search.

The following algorithm represents the methodology that we used for scheduling flexible job shops with sequence dependent setup times by using genetic algorithm.

Step 1: Initial population are created by the assignment methods. One of the three assignment methods are used while creating an initial population.

Step 2: Each chromosomes in the population are evaluated by using dispatching rules such as shortest processing time (SPT), most work remaining (MWR), longest

processing time (LPT), shortest setup time first rule (SSTFR), random rule and neighborhood search. These rules were applied for sequencing the machines.

Step 3: Two chromosomes are selected in the population by using roulette wheel selection method to apply crossover operator.

Step 4: Crossover and mutation operator are applied on the pairs of selected chromosomes.

Step 5: A new population is created by replacing a portion of the original population with the new chromosomes produced in the previous step.

Step 6: If the end condition is satisfied, stop, and return the best solution in current population, otherwise go to Step 2.

5.5 Initial Population Generation

The performance of GA is affected by various factors such as coefficients and constants, genetic operators, parameters and some strategies. Generating initial population strategies and chromosomal representations are also examples of these factors.

Better efficiency of GA can be achieved by modifying the chromosomal representations so as to generate feasible solutions, and avoiding the use of a repair mechanism. In this thesis, two chromosomal representation are used. The first one represents the machine assignment and the second one represents the operation sequence on the machines.

The initial population strategy is applied for reducing the number of search to reach the optimum design in the solution space. In this thesis, the initial population is generated by three methods and these methods are compared. The first method is based on the procedure presented in Pezella (Pezella et al., 2007) which in turn is

based on the localization approach of Kacem (Kacem et al., 2002). This approach takes into account both the processing time and the workload of the machines. The procedure consists of finding, for each operation, the machine with the minimum processing time, fixing that assignment, and then adding this time to every subsequent entry in the same column. Since this approach is strongly dependent on the order in which operations and machines, Pezella (Pezella et al., 2007), slightly modify randomly permuted jobs and machines. The second method modifies these approach taking into account the sequence dependent setup time. The third method is a random method which generates the initial population randomly. After using an assignment algorithm for finding a set of feasible schedules, they are used as an initial population for GA.

Chromosomal Representation: In this thesis, two chromosomal representations are used. The first one represents the machine assignment and the second one represents the operation sequence on the machines.

0- 1 values are used for machine assignment representation. Each unit value in the assignment representation maps a machine to a corresponding operation. In this representation, infeasible solutions are not allowed. Table 5.1 represents the processing time possibilities on various machines. Table 5.2 represents the assignment of jobs on the machines.

Table 5.1 Processing time matrix

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	3	4	1
$O_{1,2}$	3	8	2	1
$O_{1,3}$	3	5	4	7
$O_{2,1}$	4	1	1	4
$O_{2,2}$	2	3	9	3
$O_{2,3}$	9	1	2	2
$O_{3,1}$	8	6	3	5
$O_{3,2}$	4	1	8	5

Table 5.2 Machine assignment matrix

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	0	1	0
$O_{1,3}$	0	0	1	0
$O_{2,1}$	0	0	0	1
$O_{2,2}$	0	1	0	0
$O_{2,3}$	0	0	0	1
$O_{3,1}$	0	0	1	0
$O_{3,2}$	0	0	0	1

Operation sequence on the machines are represented as “operation-based representation”. In this representation, the order of operations within the permutation is interpreted as a sequence for building a schedule solution. The decoding procedure scans each permutation from left to right and uses sequence information for building a schedule consecutively. Infeasible solutions are not allowed. Each operation reads the machine data in the machine assignment matrix in Table 5.2 and the processing time data in the processing time matrix in Table 5.1. A permutation of job numbers expresses the order in which the operations of jobs are scheduled in the Gantt Chart in Figure 5.1.

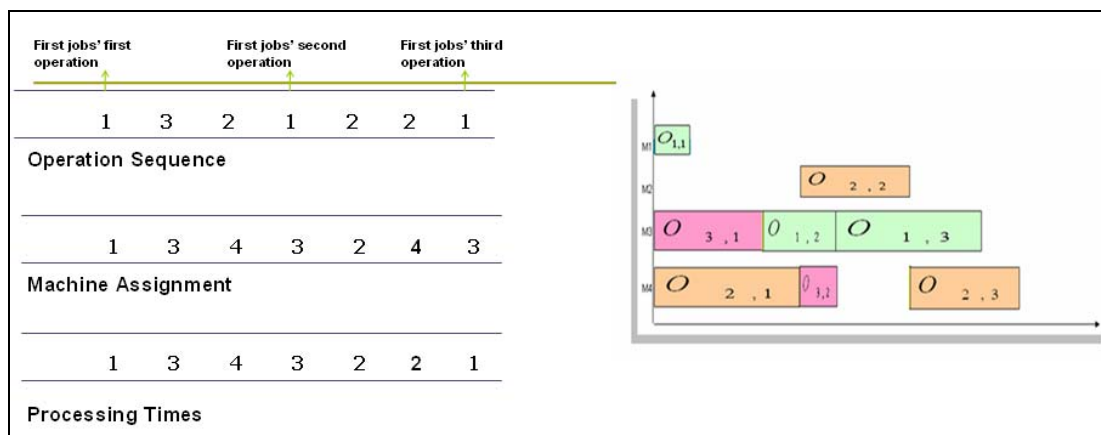


Figure 5.2: Permutation of jobs and job sequence on Gantt Chart

Approach by Localization Method: Approach by Localization method enables us to assign each operation to the suitable machine by taking into account the

processing times and the workloads of machines on which we have already assigned the operations. This method is applied for reducing the size of the search space of the problem to increase the probability of catching the global solution and enhance the performance of the GA. Pseudo code for generating initial population by Approach by Localization method is as follows:

Step 0: Create a table D , presenting the processing time possibilities for each operation on various machines.

Set $op=0$

Set $num_of_op=N$

Step 1: Randomly permute jobs and machines in the table “ D ”.

Step 2: $op=op+1$

Step 3: Find the machine with the minimum processing time for operation “ op ” and select this machine to assign operation “ op ”.

Step 4: Fixing that assignment then to add this time to every subsequent entry in the same column and update table “ D ”

Step 5: If op is equal to “ num_of_op ”?

YES, Set “ D ”

NO, Go to Step 2

The first step of the algorithm initializes a table D which represents the processing time possibilities for each operation on various machines. We define num_of_op as the number of operation and “ op ” as a counter. Second step of the algorithm, we randomly permute the jobs and the machines in the table “ D ”.

In the other steps of the algorithm, the procedure consists of finding the machine with the minimum processing time for each operation, fixing that assignment, and then adding this time to every subsequent entry in the same column.

In Approach by Localization method; first we randomly permute jobs and machines in the processing time table in Table 5.1. The assignment procedure starts with the first row. We select the machine which has the minimum processing time

for related operation. After fixing that assignment, we update the workload of the machines by adding processing time in the same column.

Table 5.3 Updated processing time matrix

	M_1	M_4	M_3	M_2		M_1	M_4	M_3	M_2
$O_{1,1}$	1	1	4	3		1	1	4	3
$O_{1,2}$	3	1	2	8		4	1	2	8
$O_{1,3}$	3	7	4	5		4	7	4	5
$O_{2,1}$	4	4	1	1		5	4	1	1
$O_{2,2}$	2	3	9	3		3	3	9	3
$O_{2,3}$	9	2	2	1		10	2	2	1
$O_{3,1}$	8	5	3	6		9	5	3	6
$O_{3,2}$	4	1	8	5		5	1	8	5

The assignment procedure continues with the second row. We select the machine which has the minimum processing time for related operation. After fixing that assignment, we update workload of the machines in Table 5.4 by adding processing time in the same column.

Table 5.4 Updated processing time matrix

	M_1	M_4	M_3	M_2	x	M_1	M_4	M_3	M_2
$O_{1,1}$	1	1	4	3		1	1	4	3
$O_{1,2}$	4	1	2	8		4	1	2	8
$O_{1,3}$	4	7	4	5		4	8	4	5
$O_{2,1}$	5	4	1	1		5	5	1	1
$O_{2,2}$	3	3	9	3		3	4	9	3
$O_{2,3}$	10	2	2	1		10	3	2	1
$O_{3,1}$	9	5	3	6		9	6	3	6
$O_{3,2}$	5	1	8	5		5	2	8	5

This procedure proceeds until all operations are assigned to a particular machine. Finally the assignment table representing the operations and their assignments to machines is generated.

Modified version of Approach by Localization: In this method, we modified the assignment procedure by using sequence dependent setup times in addition to processing times and workloads of machines.

The setup times are generated for each machine according to the a uniform discrete distribution $U[\min t_i, \max t_i]$ which t_i is the processing time of the tasks for each machine. It is assumed that the setup times between operations in machine k could not be greater than the longest processing time of operations, which requires the machine k . The setup time could not be started until the related job is free from its previous operations. In Table 5.5, the processing times of operations are presented. In Table 5.6, the setup times are generated for machine 1 according to a uniform discrete distribution $U[1, 9]$ where 1 is the minimum processing time in the Table 5.5 for machine 1 and 9 is the maximum processing time for machine 1. For each machine, setup matrices are generated by this procedure. Pseudo code for generating initial population by modified Approach by Localization method as follows:

Table 5.5 Processing time matrix

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	3	4	1
$O_{1,2}$	3	8	2	1
$O_{1,3}$	3	5	4	7
$O_{2,1}$	4	1	1	4
$O_{2,2}$	2	3	9	3
$O_{2,3}$	9	1	2	2
$O_{3,1}$	8	6	3	5
$O_{3,2}$	4	5	8	1

Table 5.6 Setup time matrix for machine 1

	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$
$O_{1,1}$	0	5	4	6	1	2	3	3
$O_{1,2}$	4	0	5	3	4	4	2	3
$O_{1,3}$	3	2	0	3	6	1	5	5
$O_{2,1}$	5	5	4	0	2	3	4	4
$O_{2,2}$	5	5	3	4	0	1	2	2
$O_{2,3}$	2	2	6	3	4	0	3	5
$O_{3,1}$	1	3	3	4	5	4	0	6
$O_{3,2}$	4	4	2	3	4	6	2	0

Step 0: Create a table “ D ”, presenting the processing time possibilities for each operation on the various machines and setup matrix “ S ” for each machine.

Set $op=0$

Set $num_of_op=N$

Step 1: Randomly permute the jobs and the machines in the table “ D ”.

Step 2: $op=op+1$

Step 3: Find the machine with the minimum processing time for operation “ op ” and select this machine to assign operation “ op ”.

Step 4: Fix that assignment, then add this time, and sequence the dependent setup time to every subsequent entry in the same column and update the table “ D ”

Step 5: If op is equal to “ num_of_op ”?

YES, Set “ D ”

NO, Go to Step 2

The first step of the algorithm initializes a table D which represents the processing time possibilities for each operation on various machines and a table S which represents the setup times of the operations. We define “ num_of_op ” as the number of operation and “ op ” as a counter. In the second step of the algorithm, we randomly permute jobs and machines in the table “ D ”.

In other steps of the algorithm, the procedure consists of finding, the machine with the minimum processing time for each operation, fixing that assignment, and then adding this time and sequence dependent setup time to every subsequent entry in the same column.

In modified version of Approach by Localization; sequence dependent setup times are used on both assignment and sequencing algorithm. First, we randomly permute the jobs and the machines in the processing time table. The assignment procedure starts with the first row. We select the machine which has the minimum processing time for related operation. After fixing that assignment, we update workload of the machines in Table 5.7 by adding processing time and the sequence dependent set up time in the same column. This procedure proceeds until all operations are assigned to a particular machine. Finally, the assignment table representing the operations and their assignments to the machines is generated.

Table 5.7 Updated processing time matrix

	M_1	M_4	M_3	M_2	M_1	M_4	M_3	M_2
$O_{1,1}$	1	1	4	3	1	1	4	3
$O_{1,2}$	3	1	2	8	3	1	2	8
$O_{1,3}$	3	7	4	5	3	7	4	5
$O_{2,1}$	4	4	1	1	4	4	1	1
$O_{2,2}$	2	3	9	3	2	3	9	3
$O_{2,3}$	9	2	2	1	9	2	2	1
$O_{3,1}$	8	5	3	6	8	5	3	6
$O_{3,2}$	4	1	8	5	4	1	8	5

Random Assignment Method: In the third method, assignments were made randomly just for the comparison purposes. Machines are randomly selected for each operation. When we assign all operations to the machines, we generate one individual. Pseudo-code for randomly generating initial population as follows:

Step 1: Create a table “ D ”, presenting the processing time possibilities on various machines

Set $op=0$

Set $num_of_op=N$

Step 2: $op=op+1$

Step 3: Randomly select one machine for operation “ op ”

Step 4: If op is equal to “ num_of_op ”?

YES, Set “ E ”

NO, Go to Step 2

The first step of the algorithm initializes a table D which represents the processing time possibilities for each operation on the various machines. We define “ num_of_op ” as the number of operation and “ op ” as a counter. In the second step of the algorithm, we randomly permute the jobs and the machines in the table “ D ”.

In the other steps of the algorithm, one operation is assigned to a specific machine randomly for each operation.

5.5.1 Fitness Evaluation

Fitness function is a particular type of objective function that prescribes the goodness of a solution in a genetic algorithm so that particular chromosome can be ranked against all the other chromosomes. In this thesis, the main objective is to minimize the makespan.

5.5.1.1 Dispatching Rules

Detailed scheduling decisions in a job shop are usually determined by dispatching rules. Most dispatching rules immediately assign the work to the machines as long as the said work is available. In this thesis, the shortest processing time, the longest processing time, the most work remaining, the first shortest setup time, and random rule are used.

Table 5.8 A list of job shop dispatch rules

<i>Rule</i>	<i>Description</i>
SPT	Select an operation with shortest processing time
MWR	Select an operation for the job with the most total processing time remaining
LPT	Select an operation with the longest processing time
SSTFR	Select an operation with the shortest setup time
RANDOM	Randomly Select an operation

5.5.1.1.1 Most Work Remaining Rule (MWR). Most work remaining rule orders the jobs in the order of most remaining processing times. Whenever a machine is freed, the job with the most remaining processing time begins processing.

5.5.1.1.2 Shortest Processing Time Rule (SPT). The shortest processing time rule sequence the jobs in the increasing order of processing times. Whenever a machine is freed, the job with the shortest processing time begins processing. In the single machine environment with zero ready time for all jobs, this rule minimizes the mean flow time and the work in process, and the mean lateness in the system.

5.5.1.1.3 Longest Processing Time Rule (LPT). The longest processing time rule sequences the jobs in decreasing order of processing times. Whenever a machine is freed, the job with the longest processing time begins processing.

5.5.1.1.4 Random Rule. Random rule orders the jobs randomly. Whenever a machine is freed, randomly selected job begins processing.

5.5.1.1.5 Shortest Setup Time First Rule (SSTFR). Shortest setup time first rule orders the jobs according to shortest setup time. Whenever a machine is freed, the job which has shortest setup time begins processing.

5.5.1.2 Neighborhood Search

A neighborhood of a schedule is defined as all possible neighbors which are generated by moving some operations in the current location to a different position. The generating mechanism is a method of taking one sequence as a seed and systematically creating a collection of related sequences. Neighborhood search algorithms which alternatively called as local search algorithms are a wide class of improvement algorithms where an improved solution may be found at each iteration by searching the “neighborhood” of the current solution. A critical issue in the design of a neighborhood search algorithm is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. As a rule of thumb, the larger the neighborhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. For this reason, a larger neighborhood does not necessarily produce a more effective heuristic unless one can search the larger neighborhood in a very efficient manner.

5.5.2 Selection

Selection is the first genetic operator. Two types of selection methods are applied in the genetic algorithms. First method is selection of reproduction, second method is selection of replacement. Selection of reproduction determines the sampling that will produce the offspring. After the production of a new individual, it is necessary to decide which individuals survive in the next generation. Selection for replacement creates a new generation from the current one and the obtained offspring (Shopova and Bancheva, 2006).

5.5.2.1 Selection of Reproduction

The selection function is used for creating an evolutionary pressure. Well performing chromosomes have a higher chance of surviving. This kind of selection

aims at identifying better chromosomes of the population that would be taken for reproduction.

In this thesis, the roulette wheel selection is used for selection of reproduction. In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness. Two individuals are then chosen randomly based on these probabilities, and the offspring is produced. The individuals of the population are assumed as slots of the roulette-wheel. Each slot is as wide as the probability for selection of corresponding chromosome is great. The scaled fitness function is used for calculating respective selection probabilities. The number of expected copies in the sampling pool is proportional to its selection probability.

5.5.2.2 Selection of Replacement

After the reproduction, we have to decide which individuals will be survived and transferred to the next generation. In this thesis, two selection of replacement methods are used. In the first method, parents replace the children randomly. Each resulting offspring will be evaluated and two individuals will be added to the population of the next generation randomly. In the second method, parents replace the children only if their children are better. Each resulting offspring will be evaluated and better one of the two individuals will be added to the population of the next generation.

5.5.3 Crossover

Crossover creates a new individual from its parents. During crossover, parents are selected from the mating pool by the roulette-wheel selection method. In this thesis, position based crossover is used. Genes from one parent's chromosome are swapped with corresponding genes on other parent's chromosome to create two children. We assign n random numbers generated from uniform distribution to each gene position of the parents, then copy the gene of parent 1 and 2 to offspring 1-2 and 2-1,

respectively. Each resulting offspring will be evaluated and better one of the two individuals will be added to the population of the next generation.

Table 5.9 Parent-1

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	0	1	0
$O_{1,3}$	0	0	1	0
$O_{2,1}$	0	0	0	1
$O_{2,2}$	0	0	0	1
$O_{2,3}$	0	1	0	0
$O_{3,1}$	0	1	0	0
$O_{3,2}$	0	1	0	0

Table 5.10 Parent-2

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	1	0	0
$O_{1,3}$	0	0	1	0
$O_{2,1}$	0	0	0	1
$O_{2,2}$	1	0	0	0
$O_{2,3}$	0	0	0	1
$O_{3,1}$	0	0	0	1
$O_{3,2}$	0	1	0	0

Table 5.11 Child-1

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	1	0	0
$O_{1,3}$	0	0	1	0
$O_{2,1}$	0	0	0	1
$O_{2,2}$	1	0	0	0
$O_{2,3}$	0	1	0	0
$O_{3,1}$	0	0	0	1
$O_{3,2}$	0	1	0	0

Table 5.12 Child-2

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	0	1	0
$O_{1,3}$	0	0	1	0
$O_{2,1}$	0	0	0	1
$O_{2,2}$	0	0	0	1
$O_{2,3}$	0	0	0	1
$O_{3,1}$	0	1	0	0
$O_{3,2}$	0	1	0	0

Two parents are selected from the mating pool (Parent-1 in Figure 5.9 and Parent-2 in Figure 5.10), n random numbers are generated from uniform distribution. Generated numbers indicate the operation located in the selected rows of those parents. Tables 5.9 and 5.10 illustrate the condition that the generated numbers are 2, 5, and 7. When child 1 and 2 are being generated from parents, selected rows of parent 1 pass only to child 2 and selected rows of parent 2 pass only to child 1. Unselected rows of parent 1 pass only to child 1 and unselected rows of parent 2 pass only to child 2. Finally Child-1 in Table 5.11 and Child-2 in Table 5.12 are generated.

5.5.4 Mutation

Mutation is a third genetic operator which is used for maintaining a genetic diversity from one generation of a population to the next generation. Individuals in the population chosen for mutation have a mutation rate. Mutation rate calculated in the following part. Mutation usually works on a single chromosome and mutation creates another chromosome or exchange the values of two string positions.

The purpose of the mutation operator is to prevent the genetic population from converging to a local minimum, and to introduce to the population new possible solutions.

1. Keep the diversity of the population in solution progress.
2. Preserve the algorithm from the premature convergence.
3. Lead for obtaining the best solutions for the reasonable number of generations.

In this thesis two mutation methods are used. First method is intelligent mutation, second method is random mutation. These mutation operators only change the assignment property of the chromosomes. In intelligent mutation; first, we select the job which has the most raised value of the effective processing time; second, we select the operation of the selected job which will mutate. Third, we select the shortest processing time from alternative machine for the selected operation. In random mutation; first, we choose the operation that will mutate. Second, we assign a new machine to this operation. Assigned machine should be chosen randomly from a set of machines that can process this operation.

Intelligent mutation reduce the effective processing time, which helps to minimize the objectives which are makespan and the total workload of the machine. Random mutation method provides the genetic diversity.

In intelligent mutation; first, we calculate effective processing time of the jobs. For example job1 has three operations ($O_{1,1}, O_{1,2}, O_{1,3}$). These operations are assigned to specific machines. Total processing times of these operations in these machines are the effective processing times of the job1. In Figure 5.13, job 1 has the most raised value of the effective processing time. Therefore, we have to cover the list of its operations to reduce this duration. Operation $O_{1,1}$, can be assigned to the machine-1 instead of the machine-3. We reduce the effective processing time to 9 units of time and the makespan to 6 units.

Table 5.13 Before intelligent mutation

	M_1	M_2	M_3	M_4	
$O_{1,1}$	0	0	1	0	} $4+2+3=9$
$O_{1,2}$	0	0	1	0	
$O_{1,3}$	1	0	0	0	
$O_{2,1}$	0	0	1	0	} $1+3+1+=5$
$O_{2,2}$	0	0	0	1	
$O_{2,3}$	0	1	0	0	
$O_{3,1}$	0	0	1	0	} $3+1=4$
$O_{3,2}$	0	0	0	1	

Table 5.14 After intelligent mutation

	M_1	M_2	M_3	M_4	
$O_{1,1}$	1	0	0	0	} $1+2+3=6$
$O_{1,2}$	0	0	1	0	
$O_{1,3}$	1	0	0	0	
$O_{2,1}$	0	0	1	0	
$O_{2,2}$	0	0	0	1	
$O_{2,3}$	0	1	0	0	
$O_{3,1}$	0	0	1	0	
$O_{3,2}$	0	0	0	1	

In random mutation, we assign n random numbers generated from a uniform (0, 1) distribution to their genes. If a random number given for a gene is less than a mutation probability then the method of random mutation is used for mutating the operation. We assign a new machine to this operation. Assigned machine should be chosen randomly from a set of machines that can process this operation.

Table 5.15 Before random mutation

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	0	0	1
$O_{1,3}$	1	0	0	0
$O_{2,1}$	0	0	1	0
$O_{2,2}$	0	0	0	1
$O_{2,3}$	0	1	0	0
$O_{3,1}$	1	0	0	0
$O_{3,2}$	0	0	0	1

Randomly select the operation

Table 5.16 After random mutation

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	0	0	0
$O_{1,2}$	0	0	0	1
$O_{1,3}$	1	0	0	0
$O_{2,1}$	0	0	1	0
$O_{2,2}$	0	0	0	1
$O_{2,3}$	0	1	0	0
$O_{3,1}$	1	0	0	0

5.6 Parameters

There are many parameters that can be considered when programming the GA procedure. These parameters are crossover rate, mutation rate, genes rate. The good performance of GA requires the proper choice of crossover and mutation operators. In this section, the adjustment method of these parameters is given. A problem is selected in initial experiments to evaluate the parameters. The considered problem is run ten times for different combinations of these parameters. The average of ten solutions is used as the performance criterion. In parameters adjustment method, at first, one of them is changed in its domain and other parameters are fixed in the

smallest amount of their domain. So the best value for variation parameter will be obtained by considering the performance criterion. This step is repeated for other parameters to find the best setting of them. The results of the preliminary experiments are shown in the following tables.

The higher the crossover and gene rate, the more quickly new structures are introduced into the population. If the crossover and gene rate is too high, high-performance structures are discarded faster than the selection can produce improvements. If the crossover and gene rate is too low, the search may stagnate due to the lower exploration rate. Current experiments allowed seven different crossover rates, varying from 0.3 to 0.9 in increments of 0.1 and the mutation rate and the gene rate are fixed at 0.05 and 0.3. The results show that the best crossover rate is 0.6.

Table 5.17 The effect of crossover rate

Crossover Rate	Mutation Rate	Gene Rate	Makespan
0,3	0,05	0,3	31.5
0,4	0,05	0,3	32
0,5	0,05	0,3	32.1
0,6	0,05	0,3	30.5
0,7	0,05	0,3	31.9
0,8	0,05	0,3	31
0,9	0,05	0,3	31.5

Eight different gene rates are allowed, which vary from 0.1 to 0.8 in increments of 0.1, and the crossover rate and the mutation rate are fixed at 0.6 and 0.05 respectively. The results show that the best gene rate is 0.3.

Table 5.18 The effect of genes rate

Gene rate	Crossover Rate	Mutation Rate	Makespan
0,1	0,6	0,05	32
0,2	0,6	0,05	31.5
0,3	0,6	0,05	30.5
0,4	0,6	0,05	31.4
0,5	0,6	0,05	31.4
0,6	0,6	0,05	32.1
0,8	0,6	0,05	30.9

Mutation is a search operator which increases the variability of the population. The current experiments allowed seven values for the mutation rate, increasing from

0.005 to 0.05 and the crossover rate and the gene rate are fixed at 0.6 and 0.3 respectively.

Table 5.19 The effect of mutation rate

Mutation Rate	Crossover Rate	Gene Rate	Makespan
0,005	0,6	0,3	31.1
0,010	0,6	0,3	31.6
0,015	0,6	0,3	32.2
0,02	0,6	0,3	32.1
0,03	0,6	0,3	31.8
0,04	0,6	0,3	31.3
0,05	0,6	0,3	30.5

In this thesis, the best values for parameters p_c , p_g and p_m are obtained as 0.6, 0.3, and 0.05, respectively. We use these parameters in the experimental design part.

5.7 Computational Results

Computational results are divided into three parts which are makespan, setup time and workload based on the size of problem instances. We use three problem instances which are small and medium sizes, and we create setup matrices for these problem instances. Setup matrices are created by the method which explained in the previous parts.

We assess the performance of genetic algorithm factors in the context of optimal solution. These factors are makespan calculation methods, initial population generation methods, selection methods, and mutation methods. Makespan calculation methods are *MWR*, *SPT*, *SSTFR*, *RANDOM*, *LPT*, *NEIGHBORHOOD*. Initial population generation methods are random generation, Approach by Localization and modified Approach by Localization method. Selection methods are best selection and random selection methods. Mutation methods are intelligent mutation and random mutation methods. Regression analysis are used for investigating the effect of the factors on the makespan, setup time and workload. Regression analysis is used for understanding which independent variables are related to the dependent variable, and for exploring the forms of these relationships. In restricted circumstances,

regression analysis can be used for inferring causal relationships between independent and dependent variables. Makespan, setup time and workload are dependent variables. Makespan calculation, initial population generation, selection, and mutation methods are independent variables.

Small-sized total flexible flexible job shop problem

In this part, we consider the factors which affect total flexible small-sized flexible job shop problems with sequence dependent setup time. The characteristics of the problem are recirculation, total flexibility and setup time. In this problem, there are 4 machines, 3 jobs, and 8 operations. Initial population size is 100 and the number of generation is 500. Parameters for this tested problem are shown in Table 5.20.

Table 5.20 Parameters for small-size test problem

Parameters	Small-sized problem
Number of jobs	3
Number of machines	4
Number of operations	8
Initial population	100
Number of generation	500

This minimization problem is solved by a combination of different factors. Each combination has four factors which are makespan, initial population, selection and, mutation methods. Total number of combination is 72. Each combination run five times, and minimum makespan, setup time and workload solutions are taken for each run. In total, 360 minimum makespan, setup time and workload solutions are obtained. Best solutions are obtained through approach by localization method, neighborhood search mehod, best selection method, and intelligent mutation method. Mean makespan, setup time and workload obtained by this combination are given in Table 5.21. Regression analysis is used for these solutions to investigate the affect of the factors on the makespan, setup time and workload. Regression analysis shows that initial population generation, selection and mutation methods affect the makespan and the workload performance. Initial population generation and selection

methods affect the setup time performance. Regression analysis results are summarized in table 5.22.

Table 5.21 GA results for total flexible small size test problem

	Mean Makespan	Mean Setup	Mean Workload
Best combination results	9	5.5	23

Table 5.22 Regression analysis results for total flexible small size test problem

	Makespan Method	Initial Population Method	Selection Method	Mutation Method
Makespan	Not effected	effected	effected	effected
Setup	Not effected	effected	effected	Not effected
Workload	Not effected	effected	effected	effected

Medium-sized partial flexible flexible job shop problem

In this part, we consider the factors which affect partial flexible medium-sized flexible job shop problems with sequence dependent setup time. The characteristics of the problem are recirculation, partial flexibility and setup time. In this problem, there are 8 machines, 8 jobs, and 27 operations. Initial population size is 100 and the number of generation is 500. Parameters for this tested problem are shown in Table 5.23.

Table 5.23 Parameters for partial flexible medium-sized test problem

Problem size	Medium
Number of jobs	8
Number of machines	8
Number of operations	27
Initial population	100
Number of generation	500

This minimization problem is solved by a combination of different factors. Each combination has four factors which are makespan, initial population, selection and mutation methods. Total number of combination is 72. Each combination run five times and minimum makespan, setup time and workload solutions are taken for each

run. Totally 360 minimum makespan, setup time and workload solutions are obtained. Best solutions are obtained through approach by localization method, neighborhood search method, best selection method, and intelligent mutation method. Mean makespan, setup time and workload obtained by this combination are given in Table 5.24. Regression analysis is used for these solutions to investigate the affect of the factors on the makespan, setup time, and workload. Regression analysis shows that population generation and selection methods affect the makespan, workload and setup time performance. Regression analysis results are summarized in table 5.25.

Table 5.24 GA results for partial flexible medium sized test problem

	Mean Makespan	Mean Setup	Mean Workload
Best combination results	35.2	77	169.8

Table 5.25 Regression analysis results for partial flexible medium sized test problem

	Makespan Method	Initial Population Method	Selection Method	Mutation Method
Makespan	Not effected	effected	effected	Not effected
Setup	Not effected	effected	effected	Not effected
Workload	Not effected	effected	effected	Not effected

Medium-sized total flexible flexible job shop problems

In this part, we consider the factors which affect total flexible medium-sized flexible job shop problems with sequence dependent setup time. The characteristics of the problem are recirculation, total flexibility, and setup time. In this problem, there are 10 machines, 10 jobs, and 30 operations. Initial population size is 100 and the number of generation is 500. Parameters for this tested problem are shown in Table 5.26.

Table 5.26 Parameters for total flexible medium-sized test problem

Problem size	Medium
Number of jobs	10
Number of machines	10
Number of operations	30
Initial population	100
Number of generation	500

This minimization problem is solved by a combination of different factors. Each combination has four factors which are makespan, initial population, selection and mutation methods. Total number of combination is 72. Each combination run five times and minimum makespan, setup time, and workload solutions are taken for each run. In total, 360 minimum makespan, setup time, and workload solutions are obtained. Best solutions are obtained through approach by localization method, neighborhood search method, best selection method, and intelligent mutation method. Mean makespan, setup time and workload obtained by this combination are given in Table 5.27. Regression analysis is used for these solutions to investigate the affect of the factors on the makespan, setup time, and workload. Regression analysis shows that population generation and selection methods affect the makespan, workload, and setup time performance. Regression analysis results are summarized in table 5.28.

Table 5.27 GA results for total flexible medium sized test problem

	Mean Makespan	Mean Setup	Mean Workload
Best combination results	29.2	136	193

Table 5.28 Regression analysis results for total flexible medium sized test problem

	Makespan Method	Initial Population Method	Selection Method	Mutation Method
Makespan	Not effected	effected	effected	Not effected
Setup	Not effected	effected	effected	Not effected
Workload	Not effected	effected	effected	Not effected

CHAPTER SIX

CONCLUSION

6.1 Conclusion

This thesis has investigated the problem of scheduling flexible job shops with sequence dependent set up time to minimize makespan objective. The flexible job shop is characterized by reentrant job shop through a number of different work centers containing identical and non-identical parallel machines. The main feature of this thesis is to combine the sequence dependent setup times with the processing times and the workloads on both assignment and sequencing procedures. The performance of genetic algorithm was also analyzed by taking into account the issues such as initial population generation, sequencing, selection, and mutation methods.

We assess the performance of genetic algorithm factors in the context of optimal solution for total flexible small-sized, total flexible medium-sized, and partial flexible medium-sized problems. These factors are makespan calculation methods, initial population generation methods, selection methods, and mutation methods. Makespan calculation methods are *MWR*, *SPT*, *SSTFR*, *RANDOM*, *LPT*, *NEIGHBORHOOD*. Initial population generation methods are random generation, Approach by Localization, and modified Approach by Localization method. Selection methods are best selection and random selection methods. Mutation methods are intelligent mutation and random mutation methods. Regression analysis are used for investigating the effect of the factors on the makespan, setup time, and workload. The genetic algorithm was developed with the purpose of solving total flexible small-sized, total flexible medium-sized, and partial flexible medium-sized problems and exploiting in comparison to different makespan calculation, initial population generation, selection, and mutation methods.

Experimental designs show that initial population generation, selection, and mutation methods affect the makespan, and workload performance for total flexible small sized problem. Initial population generation and selection methods affect the

setup time performance. For partial flexible medium sized problem, initial population generation and selection methods affect makespan, workload, and setup time performance. For total flexible medium-sized problem initial population generation and selection methods affect makespan, workload and setup time performance.

6.2 Future Research

In this thesis, we focused on flexible job shop scheduling problem with sequence dependent setup time with makespan objective. For future studies, different objectives are added such as due date, lateness, and tardiness. In this thesis, setup for a job cannot begin until the job is available for the current work center and the desired machine in the work center is idle. For future studies we may consider a setup condition in which a machine desired for a job can start setup activities immediately after being idle even though the job is not arrived to the machine yet.

REFERENCES

- Andrea R., & Gino D. (2007) Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23, 503–516
- Brandimarte P., (1993) Routing and scheduling in a flexible job shop by Taboo search. *Annals of Operations Research*, 41 (1), 157–183.
- Brucker, P., & Neyer, J. (1998) Tabu Search for the Multi-mode Job-Shop Problem. *Operations Research Spektrum*, 20, 21-28.
- Bruker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45, 369–375.
- Cheng, R., Gen, M., & Tsujimura, Y. (1996) A Tutorial Survey of Job-shop Scheduling Problems Using Genetic Algorithms, Part I: Representation. *Computers & Industrial Engineering*, 30, 983-997.
- Cheng, R., Gen, M., & Tsujimura, Y. (1999) A Tutorial Survey of Job-shop Scheduling Problems Using Genetic Algorithms, Part II: Hybrid Genetic Search Strategies. *Computers & Industrial Engineering*, 36, 343-364.
- Dauzere, Peres S., & Paulli, J. (1997) An Integrated Approach for Modeling and Solving the General Multiprocessor Job-Shop Scheduling Problem using Tabu Search. *Annals of Operations Research*, 70, 281-306.
- Fattahi, P., & Fallahi, A. (2010) Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. *CIRP Journal of Manufacturing Science and Technology*, 2, 114–123.

- Chen, H., Ihlow, J., & Lehmann, C. (1999) A genetic algorithm for flexible job-shop scheduling. *The proceedings of the 1999 IEEE international Conference on Robotics & Automation*, Detroit, 1120–1125.
- Fattahi, P., Mehrabad, M.S., & Jolai, F. (2006) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18, 331-342.
- Gao, J., Sun, L., & Gen, M. (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computer Operation Research*, 35, 2892–2907.
- Gao, J., Gen, M., Sun, L., & Zhao, X. (2008) A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers & Industrial Engineering*, 53(1), 149–162.
- Glover, F. (1990) Tabu Search – Part II. *ORSA Journal on Computing*, 2, pp.4-32.
- Hurink, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research Spektrum*, 15, 205–215.
- Joseph, Y.T. Leung, & James, H. Anderson (2004) *Handbook on Scheduling: Algorithms, Models, and Performance Analysis* CRC Press, Boca Raton, FL, USA.
- Kacem, I., Hammadi, S., & Borne, P. (2003) Approach by Localization and Multiobjective Evolutionary Optimization for Flexible Job-Shop Scheduling Problems. *IIE transactions on Systems, Man, and Cybernetics*, 32, 1-13.

- Masghouni, K., Hammadi, S., & Borne, P.(1997) Evolution Programs for Job-shop Scheduling. *Proc. IEEE International Conference on Computational and Simulation*,720- 725.
- Mason, S.J., Fowler, J.W., & Carlyle, W.M. (2002) A Modified Shift Bottleneck Heuristic for Minimizing Total Weight Tardiness in Complex Job Shops. *Journal of Scheduling*,5,247-262.
- Mason, S.J., & Oey, K. (2003) Scheduling Complex Job Shops Using Disjunctive Graphs: A Cycle Elimination Procedure. *International Journal of Production Research*,41,981-994.
- Mastrolilli, M., & Gambardella, L.M. (2000) Effective Neighbourhood Functions for the Flexible Job Shop Problem. *Journal of Scheduling*,3,3-20.
- Gen, M., Gao, J., & Lin, L.(2009) Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Intelligent and Evolutionary System SCI*,187,. 183-196.
- Ho, J.C. Tay, & E.M.K. Lai (2007) An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*,179 (2), 316–333.
- Pezzella F, Morganti G, & Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. *Computer Operation Research*, 35,3202–3212.
- Pinedo, M. (2002) *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Inc, New Jersey
- Ramiro, V., Camino, R., Vela, J.P., & Alberto, G.(2003) A Knowledge-based Evolutionary Strategy for Scheduling Problems with Bottlenecks. *European J. of Operational Research*,145 (1), 57- 71

- Shopova, E.G., & Vaklieva, G. (2006) BASIC—A genetic algorithm for engineering problems solution. *Computers and Chemical Engineering*, 30, 1293–1309.
- Xia, W.J., & Wu, Z.M. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48, 409–425.
- Valls, V., Perez, M.A., & Quintanilla, M.S. (1998) A Tabu Search Approach to Machine Scheduling. *European Journal of Operational Research*, 106, 277-300.
- Wang, W., & Brunn, P. (2000) An Effective Genetic Algorithm for Job Shop Scheduling. *Proceeding in Instn Mechanical Engineering*, 214, 293-300.
- Tay, J.C., & Ho, N. (2004) GENACE: An Efficient Cultural Algorithm for Solving the Flexible Job-Shop Problem. *Proc. IEEE, Congres of Evolutionary Computation*, 1759- 1766.

APPENDIX A: Problem Instances

Table A.1: Large Example (10 jobs, 10 machines, total flexibility)

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
$O_{1,2}$	4	1	1	3	4	8	10	4	11	4
$O_{1,3}$	3	2	5	1	5	6	9	5	10	3
$O_{2,1}$	2	10	4	5	9	8	4	15	8	4
$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
$O_{2,3}$	6	11	2	7	5	3	5	14	9	2
$O_{3,1}$	8	5	8	9	4	3	5	3	8	1
$O_{3,2}$	9	3	6	1	2	6	4	1	7	2
$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
$O_{4,1}$	5	10	6	4	9	5	1	7	1	6
$O_{4,2}$	4	2	3	8	7	4	6	9	8	4
$O_{4,3}$	7	3	12	1	6	5	8	3	5	2
$O_{5,1}$	7	10	4	5	6	3	5	15	2	6
$O_{5,2}$	5	6	3	9	8	2	8	6	1	7
$O_{5,3}$	6	1	4	1	10	4	3	11	13	9
$O_{6,1}$	8	9	10	8	4	2	7	8	3	10
$O_{6,2}$	7	3	12	5	4	3	6	9	2	15
$O_{6,3}$	4	7	3	6	3	4	1	5	1	11
$O_{7,1}$	1	7	8	3	4	9	4	13	10	7
$O_{7,2}$	3	8	1	2	3	6	11	2	13	3
$O_{7,3}$	5	4	2	1	2	1	8	14	5	7
$O_{8,1}$	5	7	11	3	2	9	8	5	12	8
$O_{8,2}$	8	3	10	7	5	13	4	6	8	4
$O_{8,3}$	6	2	13	5	4	3	5	7	9	5
$O_{9,1}$	3	9	1	3	8	1	6	7	5	4
$O_{9,2}$	4	6	2	5	7	3	1	9	6	7
$O_{9,3}$	8	5	4	8	6	1	2	3	10	12
$O_{10,1}$	4	3	1	6	7	1	2	6	20	6
$O_{10,2}$	3	1	8	1	9	4	1	4	17	15
$O_{10,3}$	9	2	4	2	3	5	2	4	10	23

Table A.2: Medium Example (8 machines, 8 jobs; partial flexibility)

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
$O_{1,1}$	5	3	5	3	3	X	10	9
$O_{1,2}$	10	X	5	8	3	9	9	6
$O_{1,3}$	X	10	X	5	6	2	4	5
$O_{2,1}$	5	7	3	9	8	X	9	X
$O_{2,2}$	X	8	5	2	6	7	10	9
$O_{2,3}$	X	10	X	5	6	4	1	7
$O_{2,4}$	10	8	9	6	4	7	X	X
$O_{3,1}$	10	X	X	7	6	5	2	4
$O_{3,2}$	X	10	6	4	8	9	10	X
$O_{3,3}$	1	4	5	6	X	10	X	7
$O_{4,1}$	3	1	6	5	9	7	8	4
$O_{4,2}$	12	11	7	8	10	5	6	9
$O_{4,3}$	4	6	2	10	3	9	5	7
$O_{5,1}$	3	6	7	8	9	X	10	X
$O_{5,2}$	10	X	7	4	9	8	6	X
$O_{5,3}$	X	9	8	7	4	2	7	X
$O_{5,4}$	11	9	X	6	7	5	3	6
$O_{6,1}$	6	7	1	4	6	9	X	10
$O_{6,2}$	11	X	9	9	9	7	6	4
$O_{6,3}$	10	5	9	10	11	X	10	X
$O_{7,1}$	5	4	2	6	7	X	10	X
$O_{7,2}$	X	9	X	9	11	9	10	5
$O_{7,3}$	X	8	9	3	8	6	X	10
$O_{8,1}$	2	8	5	9	X	4	X	10
$O_{8,2}$	7	4	7	8	9	X	10	X
$O_{8,3}$	9	9	X	8	5	6	7	1
$O_{8,4}$	9	X	3	7	1	5	8	X

Table A.3: Small Example (3 jobs, 4 machines; total flexibility)

	M_1	M_2	M_3	M_4
$O_{1,1}$	1	3	4	1
$O_{1,2}$	3	8	2	1
$O_{1,3}$	3	5	4	7
$O_{2,1}$	4	1	1	4
$O_{2,2}$	2	3	9	3
$O_{2,3}$	9	1	2	2
$O_{3,1}$	8	6	3	5
$O_{3,2}$	4	5	8	1